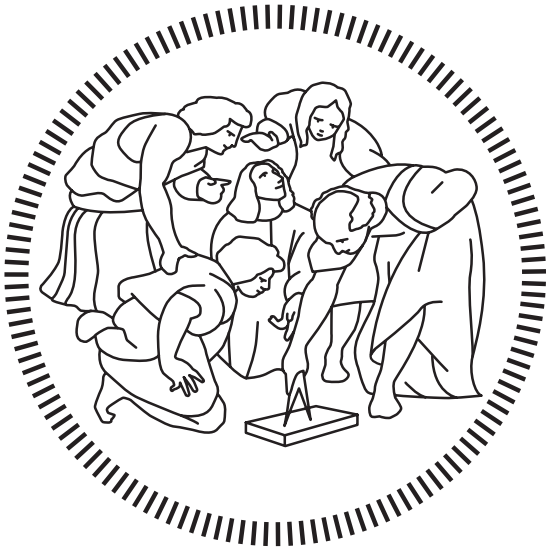
Politecnico di Milano

SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING
Master of Science – Nuclear Engineering



A new solver in OpenFOAM for the analysis of MHD problems with compressible fluids

Supervisor
Prof. Antonio CAMMI

Co-Supervisors
Dr. Stefano LORENZI
Prof. Matteo PASSONI

Candidate
Fabio VANONI – 898634

Academic Year 2019 – 2020

Ringraziamenti

Ed infine, nonostante il periodo storico un po' anomalo, sono arrivato alla conclusione del mio percorso universitario. Mi sento di affermare che, prima ancora della formazione professionale, il bene più prezioso sia quanto questa esperienza mi abbia lasciato a livello umano. Tra i muri del Politecnico e il cielo grigio della Bovisa (sempre nel cuore) sono cresciuto prima di tutto come persona e poi come ingegnere. Chiaramente, da solo non ce l'avrei mai fatta e mi sento quindi in dovere di nominare un certo numero di persone.
In primis, devo ringraziare i miei genitori, il cui supporto (morale ed economico) è stato fondamentale per il progresso della mia carriera universitaria. Insieme a loro, è d'obbligo nominare i nonni, che non mi hanno mai negato ospitalità ed affetto. Un grazie anche ai miei numerosi zii e cugini, sempre e comunque al mio fianco.
Un grande ringraziamento va poi alla mia ragazza, Letizia, che, dopo essere prepotentemente entrata nella mia vita, le ha dato una gran bella scossa. Grazie alla sua vicinanza, ho affrontato molto più facilmente gli ultimi mesi del mio percorso universitario.
Altrettanto importanti sono i miei amici storici, con cui ho condiviso tanti momenti bizzarri in tante occasioni diverse. Ringrazio quindi Andrea, Simone, Davide, Marco, Gaia, Marco, Deborah, Daniele, Sara, Gian, Marta, Denise, Dario e Laura.
Come già detto, l'università è un ambiente dove ci si forma caratterialmente, soprattutto attraverso le numerose amicizie che, per forza di cose, vengono a crearsi. Ringrazio quindi tutti gli amici che, nel corso della mia carriera, sono stati al mio fianco, sia fisicamente, durante le infinite ore di lezione, che moralmente. Prima di tutto, grazie a Francesco, Giuseppe e Giovanni, i quali, assieme al sottoscritto, sono meglio noti come gli agric0li. Un grazie anche agli altri abitanti del B12, in particolare Angelo, Riccardo, Anna Giulia, Gabriele e Alessandro.
Durante il percorso accademico, la tesi magistrale è indubbiamente la difficoltà più grande da superare, ma anche la più soddisfacente e formativa. Per questo ringrazio il professor Antonio Cammi, mio relatore di tesi, per avermi dato la possibilità di mettermi alla prova nel lavoro di tesi che segue. Un grande ringraziamento va anche al mio correlatore Stefano Lorenzi, che ha sopportato le mie frequenti intromissioni nel suo ufficio e le innumerevoli chiamate su Teams.
Infine, ritengo d'obbligo ringraziare il Magnifico Rettore Ferruccio Resta e tutti i suoi collaboratori per essere stati in grado di gestire la difficile situazione creata dal Covid-19 all'interno del sistema Politecnico. Non bisogna dare per scontata nessuna delle decisioni da loro prese, anzi ritengo che sia per buona parte grazie a loro se ora ho la possibilità di completare il mio percorso accademico.

Sommario

Il presente lavoro di tesi si propone come lo sviluppo di un nuovo solutore in OpenFOAM in grado di simulare sistemi in cui vigono le leggi della Magnetoidrodinamica (MHD) comprimibile, con la possibilità di impiegare modelli di turbolenza se necessari. La motivazione principale è da ricercare nell'assenza di tale strumento nella corrente distribuzione di OpenFOAM, mentre l'importanza degli effetti comprimibilità sui sistemi MHD è già stata studiata e confermata.
A tale scopo, si è deciso di modificare un solutore per la fluidodinamica comprimibile aggiungendoci tutti gli elementi necessari per renderlo adatto allo studio di sistemi MHD. A seguito dello sviluppo, il nuovo solutore è stato verificato con successo sul caso test noto come flusso di Hartmann, ottenendo quindi la conferma della sua capacità di trattare gli effetti magnetici.
Il secondo obiettivo è stato l'utilizzo di modelli di turbolenza in un sistema MHD, così da verificare la bontà del nuovo solutore sotto questo aspetto. Per questa ragione, si è scelto di studiare il flusso di un fluido conduttivo attraverso un canale con allargamento a gradino, mentre sotto l'effetto di un campo magnetico esterno. Una preliminare analisi di mesh sensitivity ha dato conferma del fatto che un modello di turbolenza fosse necessario per raggiungere un buon compromesso tra accuratezza nella riproduzione del campo di moto turbolento e tempo necessario allo svolgimento dei calcoli. I modelli noti come Reynolds Averaged Simulation (RAS) e Large Eddy Simulation (LES) sono stati quindi testati, ma soltanto il secondo ha restituito risultati interessanti.
Infine, è stato deciso di concludere con lo sviluppo di un modello di ordine ridotto del sistema fisico in esame, così da rendere più veloce qualsiasi tipo di sviluppo futuro del lavoro corrente. L'algoritmo noto come Dynamic Mode Decomposition è stato quindi applicato ad alcune delle simulazioni, i cui risultati sono stati ricostruiti, grazie all'algoritmo stesso, in tempi decisamente convenienti e con un buon grado di accuratezza.

Abstract

This thesis work is centered around the development of a new OpenFOAM solver which treats compressible Magnetohydrodynamics (MHD), combined with the possibility to employ turbulence models in case of need. The main motivation lies in the absence of such tool in the current OpenFOAM distribution, while the effect of compressibility on MHD systems has already been proven to be relevant.
For such purpose, it was chosen to modify a solver which treats compressible fluid dynamics by adding all the physics pertaining to MHD. It was then successfully verified on the Hartmann flow case, thus proving the accuracy of the solver in reproducing magnetic effects.
The second objective was then the employment of turbulence models in an MHD system in order to verify the reliability of the new solver under this aspect. For this reason, the flow of a conductive fluid across a backward facing step under the effect of an external magnetic field was chosen. A preliminary mesh sensitivity analysis confirmed that a turbulence model was needed in order to gain a nice compromise between accuracy in the simulation of the turbulence field and computational time. Both Reynolds Averaged Simulation (RAS) and Large Eddy Simulation (LES) models were tested, but only the latter gave interesting results.
At last, it was decided that a reduced-order model of the system could be of great help for any kind of future development. So, the Dynamic Mode Decomposition algorithm was applied to some of the most accurate simulations and, starting from that, the results were reconstructed with a good grade of precision and in a very convenient amount of time.

Estratto

Il lavoro di tesi proposto ha come obiettivo principale lo sviluppo di uno strumento in grado di simulare sistemi fisici rispondenti alle leggi della Magnetoidrodinamica (MHD) comprimibile, con il supporto di modelli di turbolenza se necessario. Tale strumento vuole essere in grado di trovare applicazione in ambiti generali, dove ad esempio la geometria può essere qualsiasi. OpenFOAM si dimostra il punto di partenza adatto in questo contesto, permettendo di sviluppare un solutore ad hoc e mantenendo quindi le caratteristiche desiderate. Oltre a tale proposito, si rende necessario lo sviluppo di un modello in grado di ridurre la dimensionalità del sistema fisico sotto esame, così da ridurre i tempi di calcolo delle simulazioni più complesse (i quali possono risultare parecchio lunghi). L'algoritmo noto come Dynamic Mode Decomposition è stato scelto come strumento adatto a tale scopo, vista in particolare l'immediatezza del suo utilizzo. In questo caso, Matlab risulta l'ambiente migliore in cui operare tale riduzione di dimensionalità, in primo luogo a causa della sua capacità di comunicazione con OpenFOAM.

Sviluppo e verifica di un nuovo solutore MHD per OpenFOAM

Nel primo capitolo, il nuovo solutore OpenFOAM viene prima scritto e in seguito verificato con un caso test. Il punto di partenza è la formulazione del problema fisico come sistema di equazioni differenziali alle derivate parziali, il quale risulta una combinazione delle equazioni della fluidodinamica comprimibile e delle equazioni di Maxwell. A seguito dell'applicazione delle tradizionali ipotesi pertinenti all'MHD (ad esempio la trascurabilità del campo elettrico), si ottiene un sistema composto dalle equazioni della fluidodinamica in cui l'accoppiamento con la parte magnetica è dato dalla forza di Lorentz nell'equazione del momento, dall'effetto Joule in quella dell'energia e dalla cosiddetta equazione dell'induzione per l'MHD:

$$\frac{\partial \vec{B}}{\partial t} = \nabla \times (\vec{u} \times \vec{B}) + \frac{\eta}{\mu_B} \Delta \vec{B} \tag{1}$$

Il sistema completo è il seguente:

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0 \\ \rho \dfrac{\partial \vec{u}}{\partial t} + \rho(\vec{u} \cdot \nabla)\vec{u} = -\nabla p + \nabla \cdot \bar{\bar{\tau}} + \rho \vec{g} + \left(\dfrac{1}{\mu_B} \nabla \times \vec{B} \right) \times \vec{B} \\ \rho \dfrac{\partial e}{\partial t} + \rho(\vec{u} \cdot \nabla)e + \rho \dfrac{\partial k}{\partial t} + \rho(\vec{u} \cdot \nabla)k = -\nabla \cdot \vec{Q} - \nabla p \cdot \vec{u} - p(\nabla \cdot \vec{u}) + \\ + (\nabla \cdot \bar{\bar{\tau}}) \cdot \vec{u} + \bar{\bar{\tau}} : \nabla \vec{u} + \rho \vec{g} \cdot \vec{u} + S_H + \dfrac{\eta}{\mu_B^2} (\nabla \times \vec{B})^2 \\ \rho = \rho(p, e) \\ \bar{\bar{\tau}} = \mu \left(\nabla \vec{u} + (\nabla \vec{u})^T \right) - \dfrac{2}{3} \mu (\nabla \cdot \vec{u}\ \bar{\bar{I}}) \\ \dfrac{\partial \vec{B}}{\partial t} = \nabla \times (\vec{u} \times \vec{B}) + \dfrac{\eta}{\mu_B} \Delta \vec{B} \end{cases} \tag{2}$$

in cui le incognite sono: la densità di massa ρ, la velocità $\vec{u}$, la pressione p, l'energia interna e ed il campo magnetico $\vec{B}$.
A questo punto è necessario implementare (2) in OpenFOAM. Il procedimento più semplice consiste nel partire da un solutore in grado di trattare i fluidi nella maniera più generale possibile ed aggiungerci l'accoppiamento con il campo magnetico. A tale scopo, è stato scelto rhoPimpleFoam, il quale tratta fluidi comprimibili utilizzando, se necessario, modelli di turbolenza quali LES e RAS. Il primo passo prevede l'aggiunta dell'equazione dell'induzione alla fine del file `rhoPimpleFoam.C`, così che venga svolto l'aggiornamento di $\vec{B}$ a seguito di quello delle altre grandezze.

```
while (bpiso.correct())
{
    fvVectorMatrix BEqn
    (
        fvm::ddt(B)
      - fvm::laplacian(DB, B)
      - fvc::curl(U ^ B)
    );

    BEqn.solve();

    #include "magneticFieldErr.H"
}
```

dove `DB` rappresenta $1/(\sigma\mu_B)$, $\sigma = 1/\eta$. Il viene poi completato con l'aggiunta della forza di Lorentz e dell'effetto Joule. Viene riportata la loro trasposizione in codice.

$$\left(\frac{1}{\mu_B} \nabla \times \vec{B} \right) \times \vec{B} \longrightarrow \texttt{DBU*(fvc::curl(B) ^ B)} \tag{3}$$

$$\frac{\eta}{\mu_B^2} (\nabla \times \vec{B})^2 \longrightarrow \texttt{ETAMU*magSqr(fvc::curl(B))} \tag{4}$$

dove `DBU` $= 1/\mu_B$ ed `ETAMU` $= \eta/\mu_B^2$.
La bontà dell'implementazione della parte magnetica deve essere ora verificata. A tal proposito, verrà utilizzato come caso test il cosiddetto flusso di Hartmann, il quale consiste nel moto di un fluido conduttivo attraverso due pareti parallele, mentre sotto l'azione di un campo magnetico esterno perpendicolare ad esse. Tale caso è stato scelto poiché permette di compiere le verifiche necessarie senza bisogno di includere effetti complessi da simulare, quali ad esempio la comprimibilità. La Figura 1 riporta una schematizzazione del caso introdotto.

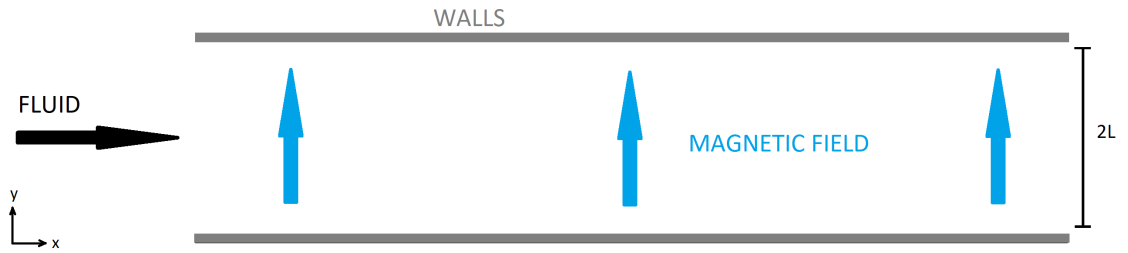


Figure 1. *Rappresentazione del flusso di Hartmann. L'orientazione degli assi x e y è riportata.*

La peculiarità del sistema fisico sotto esame è l'esistenza di un profilo analitico per quanto riguarda la componente della velocità parallela al moto:

$$u_x = u_{x,0}\frac{\cosh(Ha) - \cosh(Ha\frac{y}{L})}{\cosh(Ha) - 1} \tag{5}$$

$$Ha \equiv B_0 L\sqrt{\frac{\sigma}{\mu}} \tag{6}$$

dove $u_{x,0}$ è il valore della componente della velocità parallela al moto ed al centro del canale, L è la semi-distanza tra le pareti e B_0 è il modulo del campo magnetico esterno. Ha è il numero di Hartmann, un gruppo adimensionale importante per il sistema fisico in esame, insieme al numero di Reynolds:

$$Re \equiv \frac{\rho U L}{\mu} \tag{7}$$

Tale numero risulta importante in quanto il numero di Hartmann minimo per assicurare la validità di (5) aumenta all'aumentare del numero di Reynolds. Per questo motivo, nella verifica del solutore, $Re = 100$, così da assicurare un flusso laminare anche per valori bassi di Ha.
I risultati delle simulazioni OpenFOAM, portate a termine col nuovo solutore, sono esportati in Matlab e confrontati con il profilo dato da (5). La Tabella 1 riporta l'errore relativo integrato lungo y dei profili di u_x ad una data sezione della geometria per vari numeri di Hartmann, metre la Figura 2 mostra un confronto tra i profili ottenuti dalle simulazioni OpenFOAM ed i profili analitici.

numero di Hartmann	errore (%)
1	0.0848
5	0.0720
20	0.0555
50	0.0740

Table 1. *Errore relativo integrato lungo y dei profili di u_x ad una data sezione della geometria per vari numeri di Hartmann.*

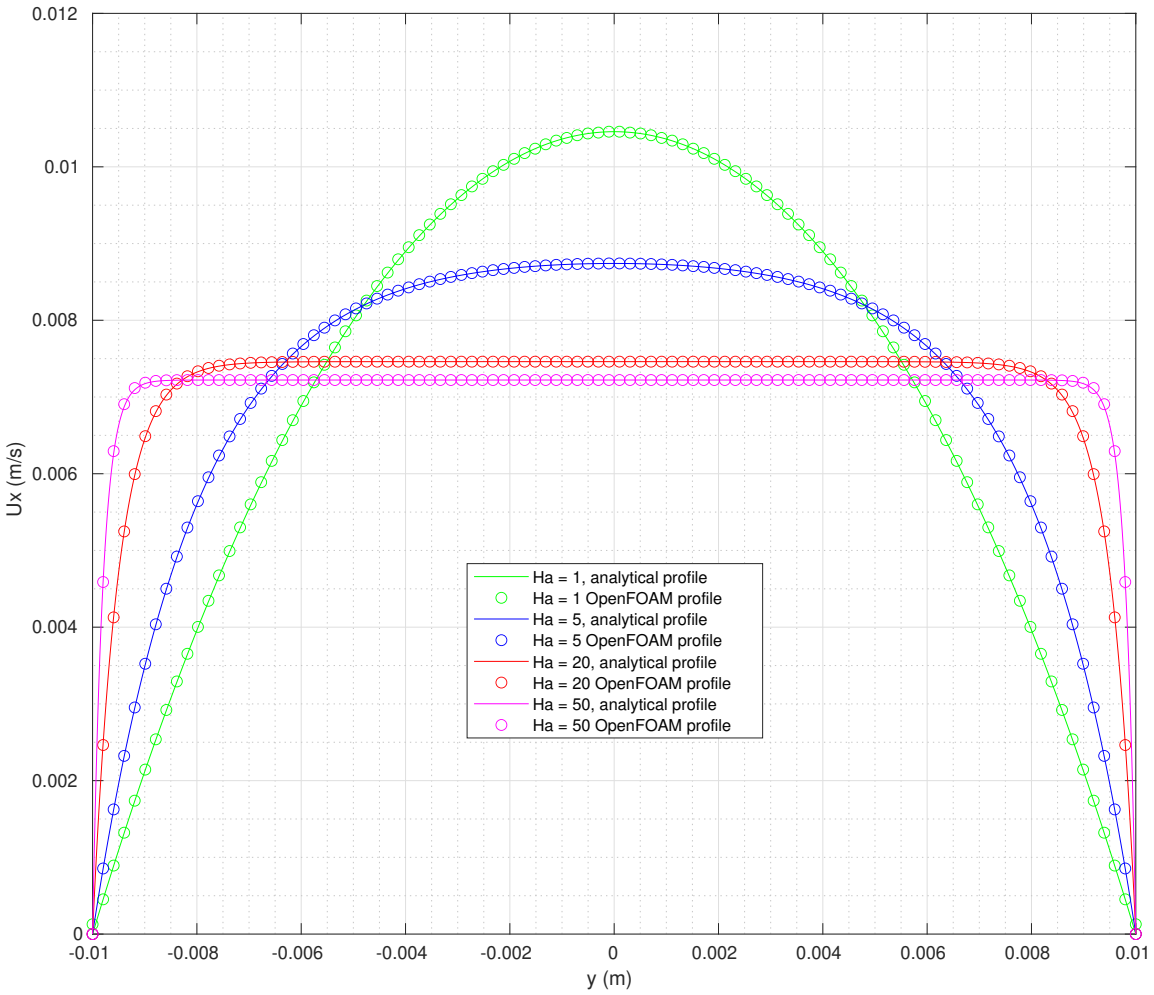


Figure 2. *Confronto tra profili analitici (linee continue) e profili OpenFOAM (punti).*

Dal grafico emerge un perfetto accordo per un ampio range di valori di Ha, risultato che conferma la bontà del nuovo solutore implementato.

Flusso MHD attraverso un allargamento a gradino

A seguito della verifica della parte magnetica, è necessario testare la capacità del solutore di applicare correttamente i modelli di turbolenza ad un caso di MHD. A tale scopo, si è optato per lo studio del moto di un fluido conduttore attraverso ad un canale formato da due pareti parallele, con una delle due che subisce un allargamento a gradino, mentre sotto l'azione di un campo magnetico esterno perpendicolare sia al moto del fluido che alle pareti stesse. Figura 3 mostra una semplice rappresentazione del caso in esame.

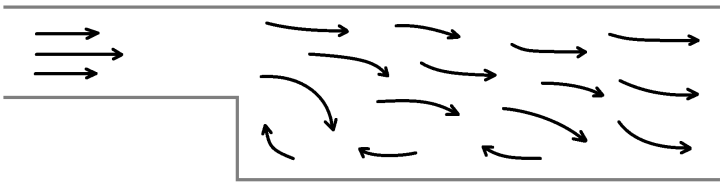
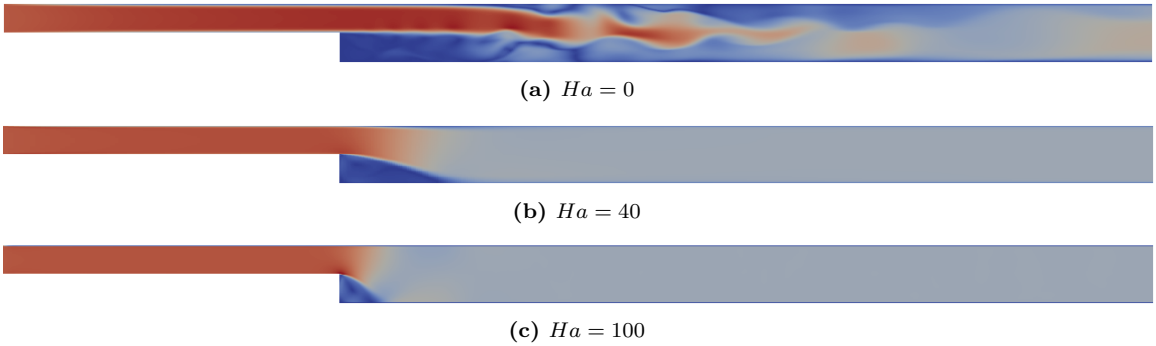


Figure 3. *Rappresentazione del flusso attraverso un allargamento a gradino. Le frecce rappresentano schematicamente il moto del fluido.*

Come studiato da [Trotta, 2019], l'effetto del campo magnetico esterno su tale sistema fisico è stabilizzante, proprietà che si traduce nella soppressione del moto turbolento. La Figura 4 mostra tale effetto.



(a) $Ha = 0$

(b) $Ha = 40$

(c) $Ha = 100$

Figure 4. *Confronto tra tre casi con valori del numero di Hatmann differenti.*

Nel caso in esame, $Re = 10000$, così da assicurare lo sviluppo di un regime di moto turbolento per il caso senza campo magnetico.
Il primo passo è lo studio della soluzione del problema al variare del livello di rifinizione della mesh, così da avere un punto di riferimento per la successiva applicazione dei modelli di turbolenza. Ne vengono impiegate 4, denominate in base al numero di elementi come: lasca, fine, molto fine ed estremamente fine. Come primo approccio, si affronta il problema come bidimensionale, ipotesi utilizzata anche da [Tano-Retamales et al., 2019], da cui sono ispirate anche le dimensioni della geometria. Tale semplificazione è accettabile e porta a risultati accurati grazie alla simmetria planare del sistema. La Figura 5 riporta il confronto tra i profili di u_x a 6 cm dal gradino ottenuti dalle diverse mesh.

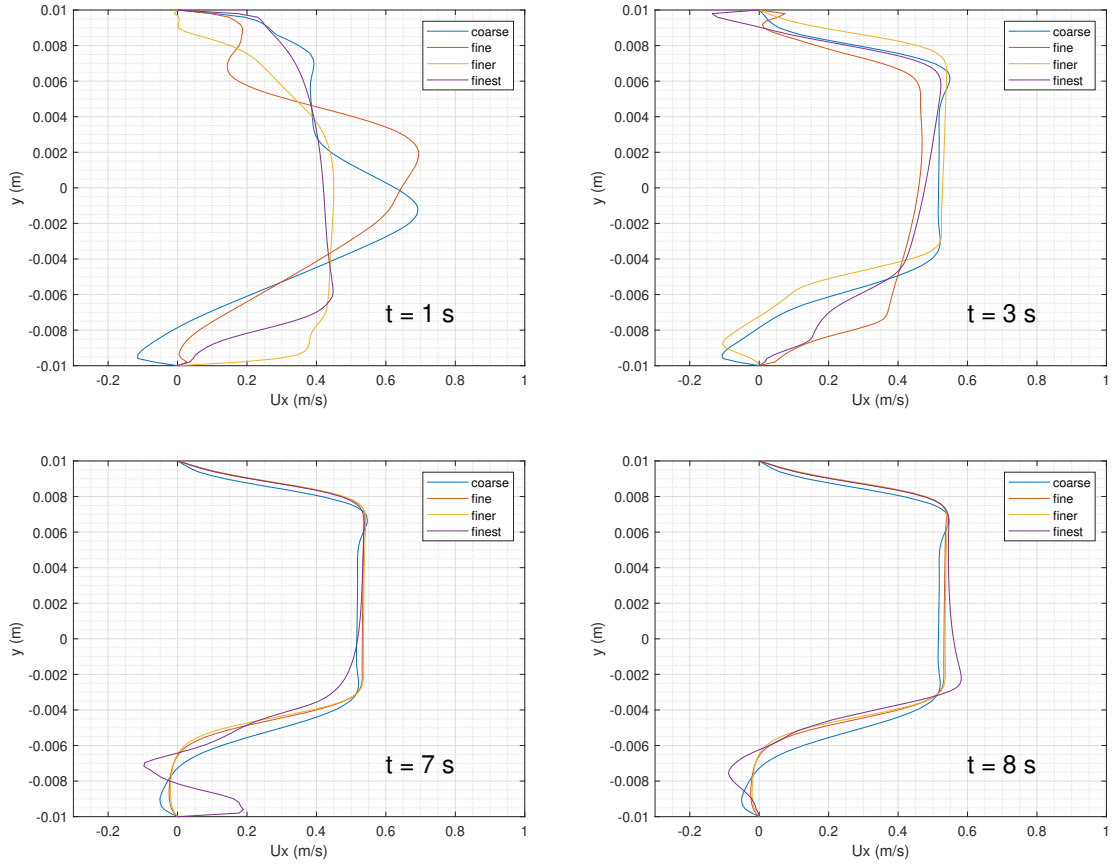


Figure 5. *Confronto tra i profili di* u_x *a* 6 cm *dallo step ottenuti dalle diverse mesh.*

Tutte le simulazioni sono state eseguite con $Re = 10000$ e $Ha = 24$. Il profilo di u_x ottenuto dalla mesh più rifinita mostra delle oscillazioni anche negli istanti finali, in cui gli altri profili sono già stabilizzati. Il caso si presta quindi all'impiego di un modello di turbolenza, così da poter simulare la presenza del campo di moto turbolento senza bisogno di raffinare eccessivamente la mesh, operazione che costa molto a livello di durata della simulazione.
L'approccio noto come Simulazione con equazioni mediate alla Reynolds (abbreviato con RAS da Reynolds Averaged Simulation) viene testato, ma con risultati mediocri. Il secondo tentativo, la Simulazione a Grandi Vortici (abbreviato con LES da Large Eddy Simulation), porta invece ad ottimi risultati. Tale modello si bassa sul filtraggio delle equazioni tramite il seguente operatore:

$$\widetilde{f}(\vec{x},t) = \int_V G(\vec{x},\vec{x}',\Delta) f(\vec{x}',t)\, d\vec{x}' \tag{8}$$

dove f è una generica funzione e G la funzione filtro. In questo modo, si può decidere di eliminare parte dell'informazione dalla soluzione (tutti i fenomeni con scale di lunghezza caratteristiche minori di Δ) mantenendo però l'influenza di tale parte sull'informazione rimanente. Questo processo ci permette di ottenere un problema per le sole grandezze filtrate, eccetto per la seguente, la quale compare nell'equazione del momento:

$$\overline{\overline{\tau}}_{SGS} \equiv \widetilde{\vec{u} \otimes \vec{u}} - \widetilde{\vec{u}} \otimes \widetilde{\vec{u}} \tag{9}$$

Tale grandezza prende il nome di tensore degli sforzi sottogriglia (SGS da sub-grid scale tensor) ed è la responsabile della conservazione dell'influenza delle scale non

risolte su quelle risolte. La modellizzazione di tale termine si può affrontare con approcci diversi: il più comune è il cosiddetto modello di Smagorinsky, ma nel caso in esame il WALE (Wall Adaptive Local Eddy-viscosity) risulta più consono.
Prima di procedere coi risultati, è importante sottolineare che le LES sono obbligatoriamente tridimensionali poiché richiedono che il campo di moto turbolento si sviluppi nel modo più realistico possibile. Tra i vari ottenuti, i risultati più interessanti riguardano: lo studio degli effetti 3D (l'importanza della componente u_z della velocità), lo studio del tempo necessario alla laminarizzazione (calcolato grazie ad uno script Matlab in grado di leggere i profili estratti da simulazioni OpenFOAM) e lo studio della lunghezza di riattacco (la distanza tra il gradino e il punto in cui la vena fluida si riattacca a parete, visibile in (b) e (c) in Figura 4). Figura 6 mostra il primo risultato, mentre Figura 7 gli altri due.

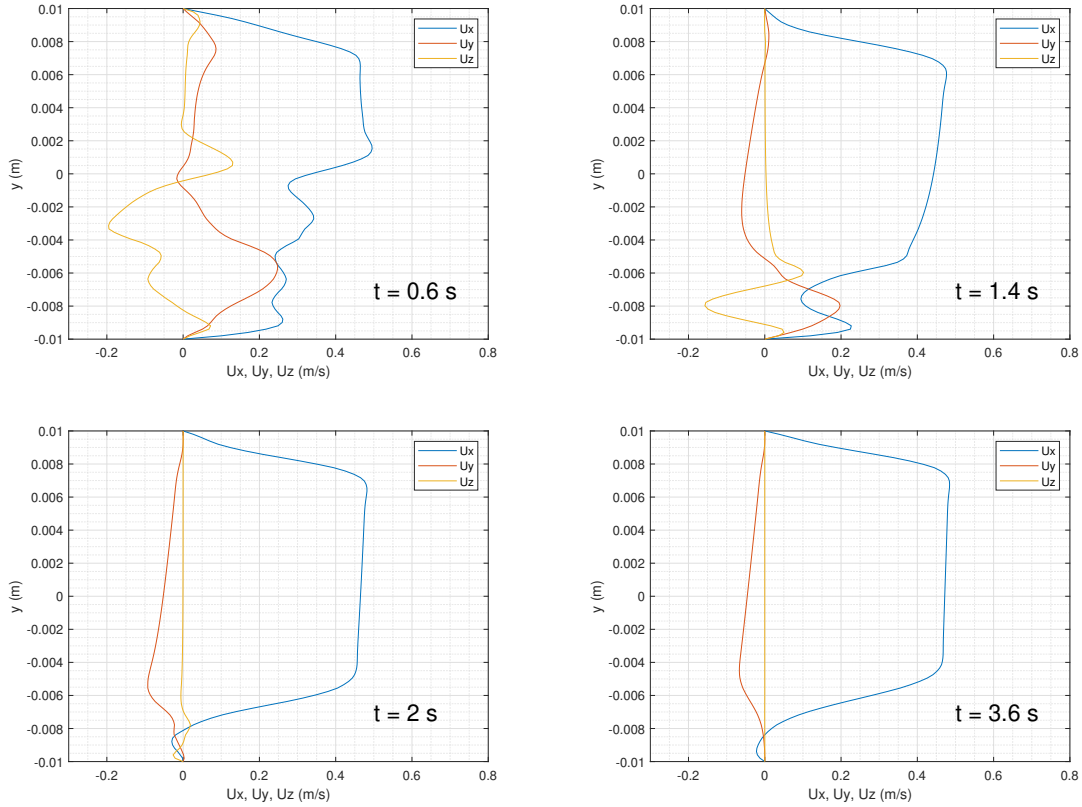
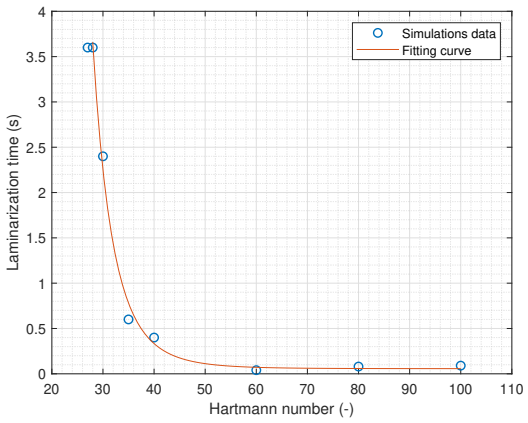
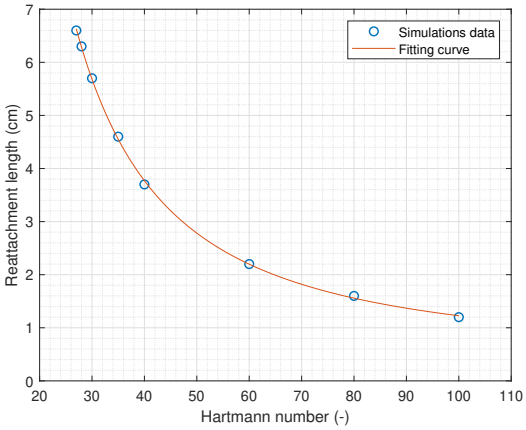


Figure 6. *Profili di u_x, u_y e u_z presi a* 6 cm *dal gradino.*

I profili sono estratti da una simulazione in cui $Re = 10000$ e $Ha = 27$. Come è possibile notare, posizionandosi sufficientemente vicini al gradino, durante il transitorio, il valore di u_z risulta paragonabile a quello delle altre componenti, per poi appiattirsi a 0 una volta raggiunto lo stato stazionario. Ciò implica che, per assicurarsi la massima accuratezza, lo studio della dinamica del sistema in oggetto è da affrontare in una geometria 3D. L'unico caso in cui il 2D può ritenersi sufficiente è nell'analisi del solo stato stazionario finale.



(a) *Ha vs tempo di laminarizzazione*



(b) *Ha vs lunghezza di riattacco*

Figure 7. *Grafici riportanti i valori ricavati dalle simulazioni OpenFOAM per il tempo di laminarizzazione e la lunghezza di riattacco, insieme alla curva interpolante, al variare del numero di Hartmann.*

Anche in questo caso, $Re = 10000$ per ogni valore riportato. L'interpolante è una legge di potenza in entrambi i casi:

$$t_l(Ha) = 9.38 * 10^{10} Ha^{-7.197} + 0.05656 \tag{10}$$

$$r(Ha) = 1124 Ha^{-1.579} + 0.4442 \tag{11}$$

Tra le due grandezze, il tempo di laminarizzazione risulta sicuramente il più delicato, in quanto la sua determinazione dipende dal tempo tra due salvataggi dei campi di moto da parte di OpenFOAM: più tale tempo diviene breve, più spazio su disco sarà occupato dai risultati, fattore che può risultare limitante.

Riduzione d'ordine tramite Dynamic Mode Decomposition

Come ultimo obiettivo, si vuole applicare l'algoritmo denominato Dynamic Mode Decomposition (DMD) ad alcune delle simulazioni portate a termine precedentemente. Il metodo prevede, in primis, la raccolta delle cosiddette snapshot del sistema (insieme di dati/misure del sistema ad un dato istante di tempo) come m vettori di stato di lunghezza n, i quali vanno a formare una matrice $n \times m$:

$$\mathbf{X}_1^m = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m\} \tag{12}$$

Ogni vettore $\mathbf{x} \in \mathbb{R}^n$, $k = 1, 2, ..., m$, corrisponde allo stato del sistema ad una distanza temporale Δt dal precedente. Si ritiene importante sottolineare che i vettori di stato possono essere sia valori estratti da simulazioni che dati raccolti da esperimenti. Nel caso in esame, gli $\mathbf{x}_k$ conterranno il campo di velocità in ogni cella della mesh ad un dato istante temporale.
A questo punto, si assume l'esistenza di una matrice $\mathbf{A}$ tale per cui

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k \tag{13}$$

In sintesi, (13) è la versione linearizzata e discretizzata in tempo dell'equazione evolutiva di un qualsivoglia sistema fisico. La DMD punta ad ottenere $\mathbf{A}$, la quale avanza lo stato del sistema di un Δt. Tale matrice rende possibile lo studio della dinamica del sistema a livello modale, oltre a permetterne previsioni future. In particolare, in questo secondo caso, l'utilizzo della DMD porterebbe ad una drastica riduzione dei tempi di calcolo poiché sarebbe possibile avanzare in tempo la soluzione senza bisogno di proseguire con lunghe simulazioni.
Il punto di partenza è l'applicazione della Singular Value Decomposition alla matrice $\mathbf{X}_1^{m-1}$:

$$\mathbf{X}_1^{m-1} \approx \mathbf{U}\mathbf{S}\mathbf{V}^* \tag{14}$$

La matrice $\mathbf{S}$ è diagonale e contiene i cosiddetti valori singolari del sistema: più uno di tali valori è grande, maggiore sarà l'importanza del suo modo associato. Poiché spesso solo alcuni di essi risultano rilevanti, si tende a selezionale solo quelli con valore maggiore ed a scartare buona parte degli altri. Deciso il numero $r < m-1$ di modi da tenere, è necessario riscalare le tre matrici $\mathbf{U}$, $\mathbf{V}$ ed $\mathbf{S}$ di conseguenza. Questa operazione ha il vantaggio di aumentare la velocità di calcolo dei passaggi successivi. A questo punto, si può ottenere $\mathbf{A}$:

$$\mathbf{A} = \mathbf{X}_2^m \mathbf{V}\mathbf{S}^{-1}\mathbf{U}^* \tag{15}$$

Poiché tale matrice può essere molto grande, nella pratica si lavora con una sua versione di ordine ridotto, data da:

$$\hat{\mathbf{A}} = \mathbf{U}^*\mathbf{A}\mathbf{U} = \mathbf{U}^*\mathbf{X}_2^m\mathbf{V}\mathbf{S}^{-1} \tag{16}$$

Tramite lo studio di autovalori e autovettori di $\hat{\mathbf{A}}$, è possibile effettuare analisi modale e ricostruzione del problema continuo, eventualmente con previsione della sua evoluzione futura.
L'algoritmo è stato implementato in Matlab, con l'obiettivo iniziale di ricostruire il campo di velocità di alcune simulazioni, così da verificare la bontà del metodo per il caso in esame. A tal proposito, è stato utilizzato la seguente formulazione di errore tra campo originale e campo ricostruito.

$$err(t) = \sqrt{\frac{\langle \mathbf{x}(t) - \mathbf{x}_{DMD}(t), \mathbf{x}(t) - \mathbf{x}_{DMD}(t) \rangle}{\langle \mathbf{x}(t), \mathbf{x}(t) \rangle}} \tag{17}$$

dove le parentesi $\langle \bullet \rangle$ indicano il prodotto scalare. In questo modo, si avrà un valore dell'errore per ogni istante di tempo.
L'algoritmo è stato applicato ad una simulazione caratterizzata da un transitorio di circa 3 secondi con l'obiettivo di studiare una delle caratteristiche più importanti del metodo: la necessità di trovare il Δt ottimale per ottenere risultati il più accurati possibile. In Figura 8 è presente il confronto tra il campo di velocità dato da una delle simulazioni (sopra per ogni istante di tempo) e la corrispondente versione ricostruita con la DMD (sotto).

(a) $t = 0.1$ s

(b) $t = 1$ s

(c) $t = 3.9$ s

Figure 8. *Confronto tra tre snapshot prese a inizio simulazione, durante il transitorio ed alla fine.*

I Δt che sono stati studiati sono 0.1, 0.05 e 0.01 s. Per ognuno di essi, tre valori di r sono stati testati. Il grafico di seguito riporta il miglior risultato per ogni Δt.

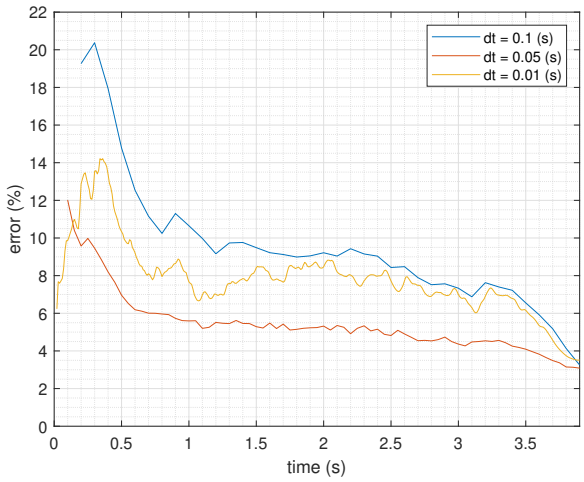


Figure 9. *Minor errore per i tre time step testati. In ordine di time step decrescente, r vale 38, 77 e 150.*

Il Δt più accurato risulta quindi 0.05 s, con un errore che decresce al di sotto del 6% prima del secondo di simulazione. Il motivo per cui 0.01 s risulta peggiore può essere dovuto al fatto che la dinamica del sistema si aggiri intorno a costanti di tempo mediamente maggiori. Un Δt troppo piccolo porta quindi ad una sovrastima dei fenomeni più rapidi e contemporaneamente ad una sottostima di quelli più lenti. Un altro motivo potrebbe risiedere nella numerica, quanto la scomposizione SVD della matrice $\mathbf{X}_1^{m-1}$ (di solito molto grande) potrebbe generare modi inesistenti nella soluzione.

Conclusioni

Nel lavoro proposto, il più importante elemento di novità introdotto è indubbiamente il nuovo solutore OpenFOAM, grazie al quale si rende possibile studiare la MHD comprimibile con il supporto di modelli di turbolenza, se necessari. Tale risultato apre la strada a simulazioni di sistemi fisici via via più complessi, grazie alla generalità introdotta dal solutore. Inoltre, il successo dell'applicazione della DMD alle simulazioni svolte apre alla possibilità di utilizzare un modello ridotto per diminuire sensibilmente i tempi di calcolo durante lo studio di un qualsivoglia sistema fisico inerente all'ambito in esame.

Contents

List of Symbols

Latin

B	magnetic field	T
c_p	specific heat capacity at constant pressure	$\mathrm{J\,kg^{-1}\,K^{-1}}$
C_S	Smagorinsky constant	
E	electric field	$\mathrm{V\,m^{-1}}$
e	specific internal energy	$\mathrm{J\,kg^{-1}}$
g	gravitational acceleration constant	$\mathrm{m\,s^{-2}}$
H	magnetic field intensity	$\mathrm{A\,m^{-1}}$
Ha	Hartmann number	
I	identity matrix	
J	electric current density	$\mathrm{A\,m^{-2}}$
k	specific kinetic energy	$\mathrm{J\,kg^{-1}}$
k_c	thermal conductivity	$\mathrm{W\,m^{-1}\,K^{-1}}$
L	characteristic length	m
l	characteristic sub-grid scale length	m
p	pressure	Pa
Pr	Prandtl number	
Q	heat flux	$\mathrm{W\,m^{-2}}$
r	reattachment length	m
Re	Reynolds number	
Re_m	magnetic Reynolds number	
S	strain rate tensor	$\mathrm{s^{-1}}$

S_H	heat source	$\mathrm{W\,m^{-3}}$
T	temperature	K
t_l	laminarization time	s
U	characteristic velocity	$\mathrm{m\,s^{-1}}$
u	velocity	$\mathrm{m\,s^{-1}}$
v	characteristic sub-grid scale velocity	$\mathrm{m\,s^{-1}}$
w	complex time frequency	$\mathrm{s^{-1}}$

Greek

α_T	turbulent thermal diffusivity	$\mathrm{m^2\,s^{-1}}$
α_{eff}	effective thermal diffusivity	$\mathrm{m^2\,s^{-1}}$
Δ	filter length	m
δ	Hartmann layer	m
δ_{ij}	Kronecker delta	
ϵ	absolute permittivity	$\mathrm{F\,m^{-1}}$
η	electrical resistivity	$\Omega\,\mathrm{m}$
κ	turbulent kinetic energy	$\mathrm{m^2\,s^{-2}}$
κ_{SGS}	sub-grid scale turbulent kinetic energy	$\mathrm{m^2\,s^{-2}}$
Λ	Kolmogorov macroscale	m
λ	Kolmogorov microscale	m
μ	dynamic viscosity	$\mathrm{Pa\,s}$
μ_B	magnetic permeability	$\mathrm{H\,m^{-1}}$
μ_T	turbulent dynamic viscosity	Pa
ν_T	turbulent kinematic viscosity	$\mathrm{m^2\,s^{-1}}$
ω	vorticity	$\mathrm{s^{-1}}$
ρ	mass density	$\mathrm{kg\,m^{-3}}$
ρ_q	electric charge density	$\mathrm{C\,m^{-3}}$
σ	electrical conductivity	$\mathrm{S\,m^{-1}}$

τ	viscous stress tensor	Pa
τ_R	Reynolds stress tensor	Pa
τ_{SGS}	sub-grid scale stress tensor	$\mathrm{m^2\,s^{-2}}$
ξ	wave number	$\mathrm{m^{-1}}$

Superscripts

$'$	fluctuating component in Reynolds decomposition
$*$	conjugate transpose
$\dagger$	Moore-Penrose pseudoinverse
$=$	double tensor
$—$	constant component in Reynolds decomposition
$\rightarrow$	vector
$\sim$	filtered quantity

List of Figures

List of Tables

Introduction

Magnetohydrodynamics (**MHD**) is the study of the properties and behaviour of electrically conducting fluids. Examples of such fluids include plasmas, liquid metals, salt water, and electrolytes. The fundamental concept behind MHD is that magnetic fields can induce currents in a moving conductive fluid, which in turn polarizes the fluid and reciprocally changes the magnetic field itself. The set of equations that describe MHD are a combination of the traditional fluid dynamics ones and Maxwell's equations. These differential equations must be solved simultaneously, either analytically or numerically. Concerning engineering problems, MHD can be found in different areas. The most famous are nuclear fusion reactors: indeed the thermonuclear plasma confinement can be studied by employing MHD. It can also find an application in the modeling of liquid-metal breeding blankets in Tokamaks, which are the most studied kind of nuclear fusion reactors. Other areas of interest are plasma propulsion in spacectraft engineering and electromagnetic casting in casting engineering.
A discrete number of works is present in the literature. [Ferroni, 2012], in his section 'State of the Art', shows a collection of all the relevant analytical results, as well as some of the numerical ones. The former are often employed as test cases for the latter: for example, among them we can find the Hartmann flow case, which will be widely discussed later on. Concerning numerical simulations, studies of MHD phenomena have been approached in many ways and with many codes, but, since we will adopt OpenFOAM as our tool[1], our interest is focused on its capabilities. In most of the past years works, its applications revolved around cases of *incompressible*[2] MHD ([Ferroni, 2012], [Tassone, 2016], [Dousset, 2009], [Woelck and Brenner, 2017]). More recently, some other fluid features have been combined with MHD: supersonic flows ([Ryakhovskiy and Schmidt, 2016], [Paudel et al., 2019]), thermal effects ([Mas de Les Valls et al., 2012]), two-fluid model ([Bodi et al., 2018]). One aspect we didn't find addressed in detail by anyone is the application of *turbulence models*. In synthesis, a turbulence model, when applied to fluid dynamics equations, allows to treat turbulence phenomena avoiding an extreme mesh refinement ([Woelck and Brenner, 2017] approaches the problem, but with a solver lacking the option to treat compressible fluid). This topic will be addressed in more detail later on.
The aim of the work is thus to develop a new solver in OpenFOAM able to treat

[1]The reason behind choosing OpenFOAM over other codes lies in its open-source nature, which gives users the freedom to modify its structure.

[2]When speaking about fluid dynamics in general, *incompressible* stands for a fluid whose mass density is considered uniform in time and space: this assumption greatly simplifies the problem by eliminating some non-linearities and couplings between the governing equations,

compressible Magnetohydrodynamics and perform simulations in any given geometry, by employing turbulence models in case of need. Two tests are then in need. The first concerns its capability to correctly treat phenomena purely linked to MHD and will be the confirmation of the correct implementation of the solver. The other will be a check of its capability to employ turbulence models efficiently in presence of MHD phenomena and will be also an opportunity to study a slightly more complex MHD system. Lastly, a dimensionality reduction algorithm, the *Dynamic Mode Decomposition*, will be applied to the simulation resulting from the previous points in order to obtain a method to save computational time for any kind of future development.

Chapter 1

Development and verification of a new MHD solver for OpenFOAM

In this chapter, the development of an OpenFOAM solver able to reproduce Magnetohydrodynamics (MHD) phenomena is explained. First of all, MHD equations are presented, underlining their differences with respect to traditional fluid dynamics. Then, one of the solvers of the 7th OpenFOAM distribution that treats compressible CFD is selected as starting point and all the required modifications applied. These revolve around the addition of an evolution equation for the magnetic field, which will be coupled with fluid dynamics through the Lorentz force and the Joule effect. The final step consists in the verification of the new solver: the *Hartmann flow* is thus presented as a potential benchmark and all the settings for the corresponding OpenFOAM simulation are explained. At last, the results are imported on Matlab, where the comparison with the analytical profile that characterizes the velocity field under the chosen benchmark conditions takes place.

1.1 Introduction

In the first chapter, the development of a new OpenFOAM solver that simulates MHD phenomena is presented. It must be said that the 7th distribution of OpenFOAM (the one used in this work) was already supplied with such solver, *mhdFoam*[1], but it only treats *incompressible* flows without the possibility to use *turbulence models* such as *Large Eddy Simulation* (LES) or *Reynolds Averaged Simulation* (RAS). Therefore, our first aim was to develop a tool which allows the user to study MHD free from any restriction on the fluid treatment and with the possibility to employ a turbulence model, if the case requires so. Such innovation can result to be fundamental in many situations: indeed, the work of [Trotta, 2019] clearly shows the impact of compressibility on the stability of MHD systems, while the need for a modelization of the turbulence field will be addressed later in this work.
Margins for innovations don't reside only in the fluid-dynamics side. In fact, most of the times in literature, the so-called *Electric Potential Method* is considered when solving MHD problems in OpenFOAM[2]. The reason we won't resort to that in the

[1] [Tassone, 2016] explained in detail how the existing solver works.
[2] Both [Ferroni, 2012] and [Tassone, 2016] introduce this approach.

present work is that it takes advantage of the following hypothesis:

$$Re_m \equiv \frac{UL\mu_B}{\eta} \ll 1 \tag{1.1}$$

where U is a characteristic velocity ($\mathrm{m\,s^{-1}}$), L is a characteristic length (m), μ_B is the *magnetic permeability* of the medium ($\mathrm{H\,m^{-1}}$) and η is the *electrical resistivity* ($\Omega\,\mathrm{m}$). Re_m is called *Magnetic Reynolds Number* and represents the ratio between advection of the magnetic field lines due to the fluid and its diffusion. If it's much lesser than one, the advection plays no role and the term $\nabla \times (\vec{u} \times \vec{B})$ in (1.14) becomes negligible, thus neglecting the coupling between $\vec{u}$ and $\vec{B}$[3]. The point is that (1.1) isn't so easily satisfied: indeed, by taking some of the numbers we will employ later in our work, precisely $U = 0.71(\mathrm{m\,s^{-1}})$, $L = 0.01(\mathrm{m})$, $\mu_B = 1.257 * 10^{-6}(\mathrm{H\,m^{-1}})$ and $\eta = 9.891 * 10^{-8}(\Omega\,\mathrm{m})$, $Re_m \approx 0.1$, which is not excessively lower than 1.
All the aforementioned reasons are necessary to choose the right path to follow in the development of the new solver, whose name will be *comprMhdFoam*. The starting point will be a solver for compressible fluid dynamics, thus ensuring no restrictions on the fluid part will be present. It will then undergo all the modifications needed to turn it into a compressible MHD solver. Thus, the right choice for the starting point will be enough to ensure an accurate treatment of all compressibility effects. What needs to be verified is only the correct implementation of the magnetic part. A simple, yet very significant case perfectly fits our requirements: the *Hartmann flow* case, which implies MHD effects with no need to consider compressibility.

1.2 The governing equations

We begin by introducing the traditional hydrodynamics equations, then proceed to couple them with Maxwell equations, in order to arrive, thanks to some simplifying hypotheses, to the MHD model[4] to be implemented in OpenFOAM.

1.2.1 Hydrodynamics equations

First of all, the traditional compressible hydrodynamics equations are presented:

- *Mass continuity equation:*

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0 \tag{1.2}$$

- *Momentum conservation equation:*

$$\rho \frac{\partial \vec{u}}{\partial t} + \rho(\vec{u} \cdot \nabla)\vec{u} = -\nabla p + \nabla \cdot \bar{\bar{\tau}} + \rho \vec{g} \tag{1.3}$$

[3]All parameters and equations will be addressed later in the development of this work, now we are only trying to give a brief motivation to our purposes. For now, it is sufficient to know that, by neglecting that term, numerical computation is simpler to accomplish.

[4]We are considering the non-relativistic case.

- *Total energy balance equation:*

$$\begin{aligned} \rho\frac{\partial e}{\partial t} + \rho(\vec{u}\cdot\nabla)e + \rho\frac{\partial k}{\partial t} + \rho(\vec{u}\cdot\nabla)k = -\nabla\cdot\vec{Q} + \\ -\nabla p\cdot\vec{u} - p(\nabla\cdot\vec{u}) + (\nabla\cdot\overline{\overline{\tau}})\cdot\vec{u} + \overline{\overline{\tau}} : \nabla\vec{u} + \rho\vec{g}\cdot\vec{u} + S_H \end{aligned} \tag{1.4}$$

- *Equation of state:*

$$\rho = \rho(p, e) \tag{1.5}$$

where ρ is the *mass density* ($\mathrm{kg\,m^{-3}}$), $\vec{u}$ is the *velocity* ($\mathrm{m\,s^{-1}}$), p is the *pressure* (Pa), $\overline{\overline{\tau}}$ is the *viscous stress tensor* (Pa), $\vec{g}$ is the *gravitational acceleration constant* ($\mathrm{m\,s^{-2}}$), e is the *specific internal energy* ($\mathrm{J\,kg^{-1}}$), k is the *specific kinetic energy*[5] ($\mathrm{J\,kg^{-1}}$), $\vec{Q}$ is the *heat flux* ($\mathrm{W\,m^{-2}}$) and S_H is a generic heat source ($\mathrm{W\,m^{-3}}$).
As first hypothesis, we will consider a *stokesian* fluid, so the viscous stress tensor will be:

$$\overline{\overline{\tau}} = \mu\left(\nabla\vec{u} + (\nabla\vec{u})^T\right) - \frac{2}{3}\mu(\nabla\cdot\vec{u}\,\overline{\overline{I}}) \tag{1.6}$$

where μ is the *dynamic viscosity* (Pa s) and $\overline{\overline{I}}$ is the identity matrix.

1.2.2 MHD equations

Maxwell's equations[6] are now introduced, before showing how they couple with hydrodynamics:

- *Gauss's Law:*

$$\nabla\cdot\vec{E} = \frac{\rho_q}{\epsilon} \tag{1.7}$$

- *Gass's law for magnetism:*

$$\nabla\cdot\vec{B} = 0 \tag{1.8}$$

- *Maxwell-Faraday equation:*

$$\nabla\times\vec{E} = -\frac{\partial\vec{B}}{\partial t} \tag{1.9}$$

- *Ampère-Maxwell equation:*

$$\nabla\times\vec{B} = \mu_B\vec{J} + \mu_B\epsilon\frac{\partial\vec{E}}{\partial t} \tag{1.10}$$

where $\vec{E}$ is the *electric field* ($\mathrm{V\,m^{-1}}$), $\vec{B}$ is the *magnetic field* (T), ρ_q is the *electric charge density* ($\mathrm{C\,m^{-3}}$), $\vec{J}$ is the *electric current density* ($\mathrm{A\,m^{-2}}$) and ϵ is the *absolute permittivity* of the medium[7] ($\mathrm{F\,m^{-1}}$).

[5] k is defined as usual: $\frac{1}{2}\vec{u}\cdot\vec{u}$.

[6] The SI units will be used.

[7] Both *permeability* and *permittivity* can be written respectively as $\mu_B = \mu_{B,0}\mu_{B,r}$ and $\epsilon = \epsilon_0\epsilon_r$, where subscript 0 refers to the value in vacuum (*vaccum permeability/permittivity*), while subscript r refers to the ratio between the value in the medium and the value in vacuum (*relative permeability/permittivity*).

The coupling occurs by adding the *Lorentz Force* to (1.3) and the *Joule Effect* to (1.4). To help the reader, they will be red-colored. Our set of equation is then:

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0 \\ \rho \dfrac{\partial \vec{u}}{\partial t} + \rho(\vec{u} \cdot \nabla)\vec{u} = -\nabla p + \nabla \cdot \bar{\bar{\tau}} + \rho \vec{g} + \rho_q \vec{E} + \vec{J} \times \vec{B} \\ \rho \dfrac{\partial e}{\partial t} + \rho(\vec{u} \cdot \nabla)e + \rho \dfrac{\partial k}{\partial t} + \rho(\vec{u} \cdot \nabla)k = -\nabla \cdot \vec{Q} - \nabla p \cdot \vec{u} - p(\nabla \cdot \vec{u}) + \\ + (\nabla \cdot \bar{\bar{\tau}}) \cdot \vec{u} + \bar{\bar{\tau}} : \nabla \vec{u} + \rho \vec{g} \cdot \vec{u} + S_H + (\vec{E} + \vec{u} \times \vec{B}) \cdot \vec{J} \\ \rho = \rho(p, e) \\ \bar{\bar{\tau}} = \mu \left(\nabla \vec{u} + (\nabla \vec{u})^T \right) - \dfrac{2}{3} \mu (\nabla \cdot \vec{u}\, \bar{\bar{I}}) \\ \nabla \cdot \vec{E} = \dfrac{\rho_q}{\epsilon} \\ \nabla \cdot \vec{B} = 0 \\ \nabla \times \vec{E} = -\dfrac{\partial \vec{B}}{\partial t} \\ \nabla \times \vec{B} = \mu_B \vec{J} + \mu_B \epsilon \dfrac{\partial \vec{E}}{\partial t} \end{cases} \tag{1.11}$$

Indeed we have a closed set of equations, but its resolution is way too costly, due to the number of equations and the various couplings between them. Some hypotheses are needed to simplify the system.
Firstly, we should consider the fact that, under MHD conditions, the fluid can be considered *quasi-neutral*, thus $\rho_q \approx 0$ [Freidberg, 2007]. Then, through a nontrivial process of combinations and substitutions between equations of (1.11), the *generalized Ohm's Law* can be obtained:

$$\vec{E} + \vec{u} \times \vec{B} = \eta \vec{J} \tag{1.12}$$

Its derivation can be found in [Chen and Chen, 2018]. Now, by taking the curl of (1.12) and (1.10)[8] and by combining them with (1.9) and (1.8), one can arrive to[9]:

$$\frac{\partial \vec{B}}{\partial t} = \nabla \times (\vec{u} \times \vec{B}) + \frac{\eta}{\mu_B} \Delta \vec{B} \tag{1.14}$$

that is the so called *Induction equation for MHD*. The quantity of variables in (1.11) is significantly reduced, since ρ_q and $\vec{E}$ are no longer to be considered. The last step consists in cutting also $\vec{J}$ out of our set by simply considering the 'MHD version' of (1.10):

$$\vec{J} = \frac{1}{\mu_B} \nabla \times \vec{B} \tag{1.15}$$

[8] In most MHD applications the *displacement current* $\epsilon \dfrac{\partial \vec{E}}{\partial t}$ in (1.10) can be neglected.

[9] The use of the following vectorial identity is required:

$$\nabla \times \nabla \times \vec{B} = \nabla(\nabla \cdot \vec{B}) - \Delta \vec{B} \tag{1.13}$$

Our final system will then be:

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0 \\ \rho \dfrac{\partial \vec{u}}{\partial t} + \rho(\vec{u} \cdot \nabla)\vec{u} = -\nabla p + \nabla \cdot \bar{\bar{\tau}} + \rho \vec{g} + \left(\dfrac{1}{\mu_B} \nabla \times \vec{B} \right) \times \vec{B} \\ \rho \dfrac{\partial e}{\partial t} + \rho(\vec{u} \cdot \nabla)e + \rho \dfrac{\partial k}{\partial t} + \rho(\vec{u} \cdot \nabla)k = -\nabla \cdot \vec{Q} - \nabla p \cdot \vec{u} - p(\nabla \cdot \vec{u}) + \\ + (\nabla \cdot \bar{\bar{\tau}}) \cdot \vec{u} + \bar{\bar{\tau}} : \nabla \vec{u} + \rho \vec{g} \cdot \vec{u} + S_H + \dfrac{\eta}{\mu_B^2} (\nabla \times \vec{B})^2 \\ \rho = \rho(p, e) \\ \bar{\bar{\tau}} = \mu \left(\nabla \vec{u} + (\nabla \vec{u})^T \right) - \dfrac{2}{3} \mu (\nabla \cdot \vec{u}\, \bar{\bar{I}}) \\ \dfrac{\partial \vec{B}}{\partial t} = \nabla \times (\vec{u} \times \vec{B}) + \dfrac{\eta}{\mu_B} \Delta \vec{B} \end{cases} \tag{1.16}$$

in the 5 variables: $[\rho \;\; \vec{u} \;\; p \;\; e \;\; \vec{B}]$[10].
System (1.16) represents compressible MHD. It is important to notice the bidirectional coupling between $\vec{u}$ and $\vec{B}$, both from the physical and numerical point of view. Physically, the fluid will feel the presence of an *externally applied magnetic field* due to the Lorentz force, but then $\vec{B}$ itself will react to the change in $\vec{u}$, so an *induced magnetic field* will arise and superimpose itself to the external one[11]. Numerically, the coupling is reflected in the proper arrangement of the time step, as discussed in the next section.

1.3 The OpenFOAM code

Now that we introduced the physics of MHD, the implementation of the code can be approached. Since the 7th distribution of OpenFOAM provides a large amount of solvers, an accurate analysis of all the options was necessary. In conclusion, the best starting point turned out to be *rhoPimpleFoam*. As already mentioned, it treats *compressible* flows, with the help of *turbulence models* if needed. Moreover, it can solve most fluid dynamics scenarios, like flows with buoyancy effects, heat transfer, transonic behavior and so on. One other key characteristic of rhoPimpleFoam is its capability to operate in *transient mode*, i. e., to solve time-dependent problems, thus allowing to study the dynamics of the system. Its name derives from the algorithm it employs to calculate fields, PIMPLE, which algorithm is reported in Figure 1.1.

[10] A model for $\vec{Q}$ is necessary to close the system. Later in this work, precisely in the code development section, we will see how OpenFOAM treats it.

[11] $\vec{B}$ can be defined as:

$$\vec{B} = \vec{B_0} + \mu_B \vec{H} \tag{1.17}$$

where $\vec{B_0}$ is the *externally applied magnetic field* (T) and $\vec{H}$ is the *magnetic field intensity* ($\mathrm{A\,m^{-1}}$). This definition reflects what just discussed regarding the induction of $\vec{B}$, but it is not necessary in our dissertation since, as we will see, it would only introduce unnecessary complications in the code.

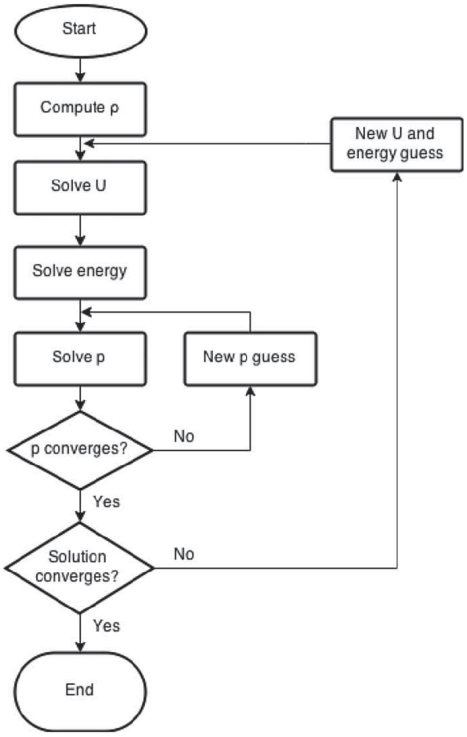


Figure 1.1. *PIMPLE algorithm reported as flux diagram*

1.3.1 Induction Equation of MHD

The first step is the addition of (1.14) at the bottom of the file `rhoPimpleFoam.C`, inside the `while (runTime.run())` cycle:

```
while (bpiso.correct())
{
    fvVectorMatrix BEqn
    (
        fvm::ddt(B)
      - fvm::laplacian(DB, B)
      - fvc::curl(U ^ B)
    );

    BEqn.solve();

    #include "magneticFieldErr.H"
}
```

where `DB` is defined as $1/(\sigma\mu_B)$[12]. The syntax of the differential operators is straightforward and the $= 0$ is implied. What is less obvious is the meaning of `fvm` and `fvc`: they stand, respectively, for *finite volume method* and *finite volume calculus* and define whether a term will be treated *implicitly* (`fvm`) or *explicitly*[13](`fvc`) while

[12] $\sigma = 1/\eta$ is called *electrical conductivity* ($\mathrm{S\,m^{-1}}$). It was employed in the code instead of η.

[13] An *implicit* term is treated as a variable while solving for a given time step, while an *explicit* one considers the value of variables at the previous time step and is therefore known.

solving the equation. This is a particularly important aspect because it is strictly related to how our system (1.16) will be solved: in order to speed up the calculations, OpenFOAM is programmed in such a way to consider every equation by itself, in the so-called *segregated* way. This implies that, when solving a time step, every equation must contain only one variable, with all the other quantities considered explicitly. So we just motivated why `curl(U ^ B)` is preceded by `fvc`: it contains `U` while having `B` as variable.
Apart from the equation syntax, other aspects must be briefly discussed. One is the row `#include "magneticFieldErr.H"`, which allows to check if $\nabla \cdot \vec{B}$ is small enough to be considered 0 after every iteration of our equation, which number is controlled by the user and assured by the `while (bpiso.correct())` cycle. This is the last aspect to be clarified: thanks to it one can ask OpenFOAM to solve the equation more than once in succession, while updating the explicit term every time with the just calculated value of `B`.

1.3.2 Momentum conservation equation

The addition of the *Lorentz Force* to (1.3) is now performed. Its transposition into code language is as follow:

$$\left(\frac{1}{\mu_B}\nabla \times \vec{B}\right) \times \vec{B} \longrightarrow \texttt{DBU*(fvc::curl(B) ^ B)} \tag{1.18}$$

where $\texttt{DBU} = 1/\mu_B$. The original equation can be found inside the file `UEqn.H` in the aforementioned directory. We will report the modified version:

```
MRF.correctBoundaryVelocity(U);

tmp<fvVectorMatrix> tUEqn
(
    fvm::ddt(rho, U)
  + fvm::div(phi, U)
  - DBU*(fvc::curl(B) ^ B)
  + MRF.DDt(rho, U)
  + turbulence->divDevRhoReff(U)
 ==
    fvOptions(rho, U)
);
fvVectorMatrix& UEqn = tUEqn.ref();

UEqn.relax();

fvOptions.constrain(UEqn);

if (pimple.momentumPredictor())
{
    solve(UEqn == -fvc::grad(p));
```

```
    fvOptions.correct(U);
    K = 0.5*magSqr(U);
}
```

The only addition here is the already mentioned term. We are not interested in discussing in detail all the syntax, but only in underlining some aspects that we consider more relevant. Firstly, we want to motivate at least the first two terms of the equation, since there is some maths behind them. Indeed if one makes good use of (1.2), the following equality holds:

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} = \frac{\partial(\rho\vec{u})}{\partial t} + \nabla \cdot (\rho\vec{u} \otimes \vec{u}) \tag{1.19}$$

OpenFOAM utilizes the right hand side formulation, with ϕ representing the product $\rho\vec{u}$. Secondly, the entry `fvOptions(rho, U)` is present in the right hand side, allowing us to insert forcing terms of various nature directly from the case setup[14](the buoyancy term can be inserted in this way, it is not present by default). It's important to notice that both our new term and `grad(p)` are treated explicitly, while `U` is the variable, accordingly to what previously discussed regarding the solution of equations. Lastly, it is important to introduce the entry `UEqn.relax()`, since it represents a very useful feature of our starting solver: the application of *relaxation factors* to the current variable between consecutive iterations[15].

1.3.3 Total Energy balance equation

Finally, the last equation, (1.4), found inside `EEqn.H`. Here, the *Joule Effect* must be added, after its trasposition:

$$\frac{\eta}{\mu_B^2}(\nabla \times \vec{B})^2 \longrightarrow \texttt{ETAMU*magSqr(fvc::curl(B))} \tag{1.20}$$

where $\texttt{ETAMU} = \eta/\mu_B^2$. The file reads:

```
{
    volScalarField& he = thermo.he();

    fvScalarMatrix EEqn
    (
```

[14]The case setup is the definition of all the conditions and parameters of a simulation. It will be discussed later, but we can anticipate that fvOptions won't be used in our case. since buoyancy doesn't play an important role in the Hartmann case scenario.

[15]*Relaxation factors* control *under-relaxation*, a technique used for improving stability of a computation. An *under-relaxation factor* α, $0 < \alpha \leq 1$, specifies the amount of under-relaxation, as described below:

- $\alpha = 1$ means no under-relaxation.
- α decreases, under-relaxation increases.
- $\alpha = 0$ implies that solution doesn't change between iterations.

An optimum choice of α is one that is small enough to ensure stable computation but large enough to move the iterative process forward quickly.

```
        fvm::ddt(rho, he) + fvm::div(phi, he)
      + fvc::ddt(rho, K) + fvc::div(phi, K)
      + (
            he.name() == "e"
          ? fvc::div
            (
                fvc::absolute(phi/fvc::interpolate(rho), U),
                p,
                "div(phiv,p)"
            )
          : -dpdt
        )
      - fvm::laplacian(turbulence->alphaEff(), he)
      - ETAMU*magSqr(fvc::curl(B))
     ==
        fvOptions(rho, he)
    );

    EEqn.relax();

    fvOptions.constrain(EEqn);

    EEqn.solve();

    fvOptions.correct(he);

    thermo.correct();
}
```

As for the previous one, we don't need to deal with the explanation of all the code, but just point out the most important aspects. At first, we should consider the fact that, with respect to (1.4), the terms related to $\overline{\overline{\tau}}$, the buoyancy and the heat source are excluded by default, though it's possible to add them thanks to the `fvOptions(rho, he)` entry[16]. Secondly, the heat flux is described by the following:

$$\vec{Q} = -\alpha_{eff}\nabla e \implies \nabla \cdot \vec{Q} = -\alpha_{eff}\Delta e \tag{1.21}$$

where α_{eff} is the *effective thermal diffusivity* ($\mathrm{m^2\,s^{-1}}$), which is the sum of both laminar and turbulent thermal diffusivities. This is fundamental to close the system of equation as mentioned before in footnote 10.

1.3.4 Declaration of $\vec{B}$ and new fluid properties

Finally, the magnetic field must be declared as a new variable of our problem, together with μ_B and σ. In order to do so, we need to add some lines of code to the file `createFields.H`:

[16] As for $\overline{\overline{\tau}}$ terms, the *rhoCentralFoam* solver implements them into the total energy balance, so their syntax could be copied directly from there into our equation.

```
volVectorField B
(
    IOobject
    (
        "B",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

#include "createPhiB.H"
```

Here we declared `B`. Being it a vector field, its declaration is the same as `U`. We also included the file `createPhiB.H`, which is required in order to calculate the error on $\nabla \cdot \vec{B}$. It can be found in the directory reported at the end of 1.3.2.

```
IOdictionary transportProperties
(
    IOobject
    (
        "transportProperties",
        runTime.constant(),
        mesh,
        IOobject::MUST_READ_IF_MODIFIED,
        IOobject::NO_WRITE
    )
);
```

The last piece of code allows us to add the *transportProperties* dictionary to a case setup, through which we can specify values for μ_B and η. But first, we need to declare them:

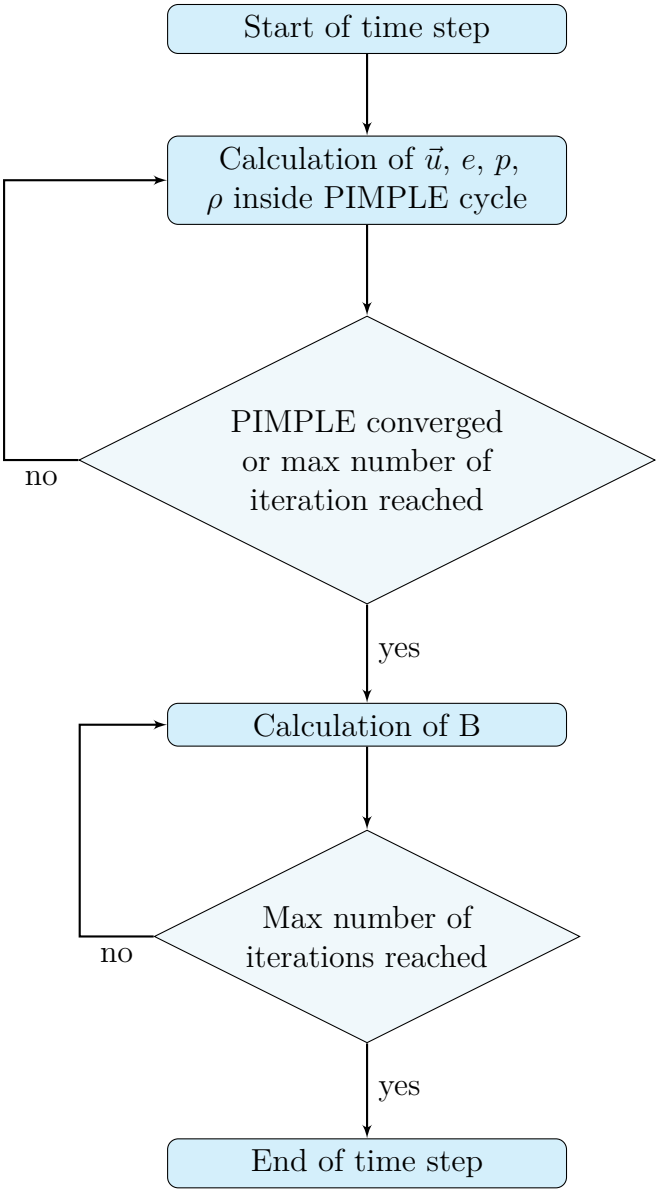
```
dimensionedScalar muB
(
    "muB",
    dimensionSet(1, 1, 0, 0, 0, -2, 0),
    transportProperties
);

dimensionedScalar sigma
(
    "sigma",
    dimensionSet(-1, -3, 1, 0, 0, 2, 0),
    transportProperties
);
```

```
dimensionedScalar ETAMU = 1.0 /(muB*muB*sigma);
ETAMU.name() = "ETAMU";

dimensionedScalar DBU = 1.0/(muB);
DBU.name() = "DBU";
```

Here also `ETAMU` and `DBU` are declared. As already mentioned in footnote 12, we didn't use *electrical resistivity* in the code, but *electrical conductivity* instead.
In conclusion, we will report a simple flux diagram showing how all fields are calculated in a time step.



1.4 Hartmann flow case

1.4.1 Case presentation

In order to verify the goodness of the newly developed solver, we chose the *Hartmann flow*, which is the flow of a conductive fluid between two infinitely extended, parallel plates, while exposed to an external magnetic field perpendicular to the direction of motion. Figure 1.2 shows geometry and fields directions.

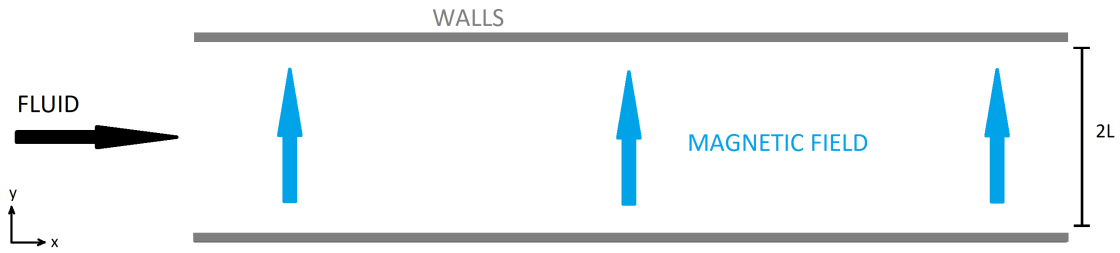


Figure 1.2. *Hartmann flow showcase. Axes x and y orientation is shown.*

Our interest in this configuration comes from the fact that, if the intensity of $\vec{B}$ is high enough, the flow will become laminar and the component of the velocity field parallel to the direction of motion will be described by the following analytical profile:

$$u_x = u_{x,0}\frac{\cosh(Ha) - \cosh(Ha\frac{y}{L})}{\cosh(Ha) - 1} \tag{1.22}$$

where $u_{x,0}$ is the value of the component of velocity parallel to the motion at the center of the channel, L is the half-distance between plates and Ha is the *Hartmann number*, an adimensional group defined as:

$$Ha \equiv B_0 L\sqrt{\frac{\sigma}{\mu}} \tag{1.23}$$

where B_0 is the module of the external magnetic field. The Hartmann number is one of the two relevant adimensional group of this problem, the other being the *Reynolds number*:

$$Re \equiv \frac{\rho U L}{\mu} \tag{1.24}$$

where U is a characteristic velocity (for us, it will be its inlet value)[17]. Both numbers must be considered because the minimum Ha that guarantees laminarization of the motion is influenced by how much turbulent the flow originally is, so the higher the Re, the higher the minimum Ha. Moreover, if Ha and Re are set so that laminarization will surely occur, the problem can be solved as two-dimensional, as for a plane Poiseuille flow. For this reason, Re will be very low, thus ensuring laminar motion for every Ha value. Finally, if no thermal gradients are present, the case can be treated

[17]Usually, the characteristic length used in defining Re is the *hydraulic diameter*. Here we opted for L instead in order to be coherent with Ha definition.

as incompressible MHD, which is ideal for our purpose[18].
One last important aspect to underline is that walls are made out of *insulating* material. Indeed if we consider $\sigma_{wall} > 0$ we aren't dealing with the Hartmann case scenario anymore, since u_x will not be described by (1.22)[19].

1.4.2 OpenFOAM case setup

The case structure of our problem is now introduced: it consists of 3 directories, each one containing dfferent files.

```
<case directory name>
|
|---0
|   |---B
|   |---p
|   |---T
|   |---U
|
|---constant
|   |---thermophysicalProperties
|   |---transportProperties
|   |---turbulenceProperties
|
|---system
    |---blockMeshDict
    |---controlDict
    |---decomposeParDict
    |---fvSchemes
    |---fvSolution
    |---singleGraph-x
```

0/B

This directory contains one file for every variable, each one specifying initial and boundary condition for that field, except for ρ. In fact, as we will see, it will be calculated from the equation of state specified in the `thermophysicalProperties` file. For the sake of clarity, the geometry we will adopt is firstly introduced: our domain will be a simple rectangle, since we will approximate the case as two-dimensional. It is also important to specify a name for every one of its boundaries because they will appear into the files mentioned above. Figure 1.3 shows the name associated to each side.

[18] Both the 2D and the incompressible nature of the case are characteristics which allows for faster simulations. Since the purpose is to test the goodness of the magnetic field implementation, they are surely welcome.

[19] [Ferroni, 2012] in 3.1 lists some theoretical results for which $\sigma_{wall} > 0$. In those cases, the analytica profile is different from the one in (1.22).

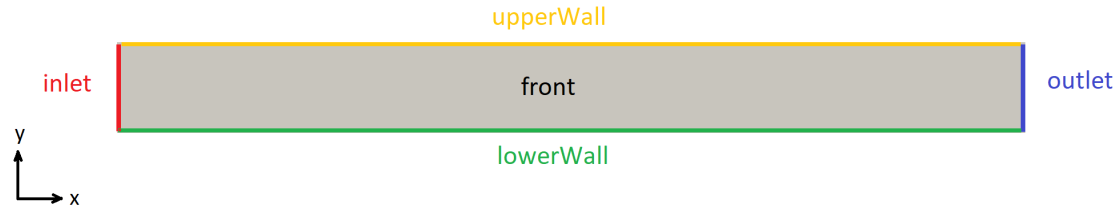


Figure 1.3. *Geometry presentation. The names of every boundary are reported*

The reason why also `front` is named will be soon explained. It's important to specify that our domain won't include walls, but only the space between them.
We will now introduce `B`. It will be also our starting point to explain some general features of OpenFOAM case files.

```
dimensions      [1 0 -1 0 0 -1 0];

internalField   uniform (0 B_0 0);

boundaryField
{
    inlet
    {
        type            zeroGradient;
    }

    outlet
    {
        type            zeroGradient;
    }

    lowerWall
    {
        type            fixedValue;
        value           uniform (0 B_0 0);
    }

    upperWall
    {
        type            fixedValue;
        value           uniform (0 B_0 0);
    }

    frontAndBack
    {
        type            empty;
    }
}
```

where `B_0` is the same value B_0 that appears in the definition of Ha. This means that

its value is set basing on the desired value for the adimensional group. As previously mentioned, all boundary names appear, together with the applied type of boundary conditions. Here two of them are employed:

- `fixedValue` allows us to impose a field value on that boundary that won't be influenced by the physics of the problem. The entry `uniform` specifies that this value will be constant in space[20]. Finally, a vector is needed to assign the desired numerical value, since we are dealing with a vector field.

- `zeroGradient` means that the following condition is imposed:

$$\frac{\partial \vec{B}}{\partial n} = 0 \tag{1.25}$$

  where n specifies the direction normal to the boundary. This means that the field will remain costant in space along n direction.

We didn't include the `empty` type because it isn't a real boundary condition, but rather the way to inform OpenFOAM that we want a two-dimensional simulation: by imposing that on the front and the back face of our geometry, OpenFOAM will automatically set the z components of our vector field to 0, without the need to calculate them.
Regarding the initial condition, it is specified by the voice `internalField`, using the same sytax of `fixedValue`.
One last comment goes to the first line, where the unit of measurement of our field must be specified. In order to give dimensions to all quantities, OpenFOAM requires the user to write a vector of seven numbers, each one corresponding to the exponent of one of the SI base unit, arranged in the following order: mass (kg), length (m), time (s), thermodynamic temperature (K), amount of substance (mol), electric current (A) and luminous intensity (cd). So, since $(\mathrm{T}) = (\mathrm{kg\,s^{-1}\,A^{-1}})$, the correct entry in the case of the magnetic field will be `[1 0 -1 0 0 -1 0]`.
Before proceeding to the next field, it is mandatory to expose the physical motivations behind initial and boundary conditions of `B`. At time 0, when the fluid is still, its value will be the same in all the domain, walls included(we can imagine the field generated by two magnets outside the walls). When the fluid starts moving, the field will induce a current in it, which, in response, will induce a magnetic field that superimposes itself to the external one. This results in a variation of `B` inside the domain, but not on walls, where the value is imposed. The physical translation of our choice is the imposition of $\sigma_{wall} = 0$: indeed if walls had non-null conductivity, the current generated inside the fluid would enter them, inducing a magnetic field. At that point, our boundary condition wouldn't be valid anymore. Since OpenFOAM is not yet provided with the right tools to deal with such case[21], the Hartmann flow is the optimal benchmark for our solver, being it easily reproducible with our tools. Regarding the conditions on inlet and outlet, `zeroGradient` is the right choice because there is no physical reason for `B` to have a non-null gradient along these directions.

[20] OpenFOAM allows also to impose profiles along a boundary, like a parabolic inlet for the velocity.
[21] The implementation of a new boundary condition in OpenFOAM is a problem that goes beyond the scope of this thesis work, since it requires a deep knowledge of the source code.

0/U

`U` is now introduced:

```
dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform (0.0071 0 0);
    }

    outlet
    {
        type            zeroGradient;
    }

    upperWall
    {
        type            noSlip;
    }

    lowerWall
    {
        type            noSlip;
    }

    frontAndBack
    {
        type            empty;
    }
}
```

Here the condition `noSlip` appears. Its purpose is straightforward: it simply imposes all components of velocity to be uniformly 0 on that boundary. Regarding the other conditions, we need to set a `fixedVaue` at the inlet in order to control Re, since the reported value represents U inside (1.24). Considering its definition, we will have $Re \approx 100$, so that also low values of Ha will ensure the validity of (1.22) (ρ, L and μ values will be addressed later). As for the outlet, `zeroGradient` is the optimal choice, since there is no physical reason for `U` to have a gradient in this direction.
Finally, the initial condition is uniformly 0 for every component. It is usually the standard choice in case of transient simulations and implies that a minimum amount of time is required before the profile will correctly develop.

0/p

p is next:

```
dimensions      [1 -1 -2 0 0 0 0];

internalField   uniform 100000;

boundaryField
{
    inlet
    {
        type            zeroGradient;
    }

    outlet
    {
        type            fixedValue;
        value           uniform 100000;
    }

    upperWall
    {
        type            zeroGradient;
    }

    lowerWall
    {
        type            zeroGradient;
    }

    frontAndBack
    {
        type            empty;
    }
}
```

Boundary conditions here are mostly dictated by numerics: when velocity is imposed at the inlet, it is preferable to impose pressure at the outlet, with `zeroGradient` on all other boundaries. By setting the outlet value also as initial condition, simulation stability is assured. Since we are not dealing with fluids under pressure, $p = 100000$Pa is optimal.

0/T

Finally, `T`:

```
dimensions      [0 0 0 1 0 0 0];

internalField   uniform 380;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 380;
    }

    outlet
    {
        type            zeroGradient;
    }

    lowerWall
    {
        type            zeroGradient;
    }

    upperWall
    {
        type            zeroGradient;
    }

    frontAndBack
    {
        type            empty;
    }
}
```

To guarantee numerical stability, it is better to impose a temperature on the same boundary as velocity, i. e., inlet, plus an initial condition different from 0, which for us should be set equal to the inlet one. The reason behind the chosen value is that we will adopt *liquid sodium* as conductive fluid, so we need $T \geq T_{melt} \approx 371(\text{K})$.
Since we won't deal with thermal effects in our benchmark, `zeroGradient` is the perfect choice for all other boundaries. Indeed a condition like (1.25) for T has the physical meaning of *adiabatic wall*, thus ensuring that no heat transfer phenomena will be present and the incompressibility of the case is maintained.
Finally, it is mandatory to show the relation used by OpenFOAM to switch from temperature to specific internal energy:

$$e = c_p T \tag{1.26}$$

where c_p is the *specific heat capacity at constant pressure* ($\mathrm{J\,kg^{-1}\,K^{-1}}$).

constant/thermophysicalProperties

Firstly, We will briefly present `constant`, the directory in which the physical parameters are set, together with other settings for energy and state equations. Usually, an OpenFOAM case folder doesn't contain a `transportProperties` file if it treats compressible fluids, but since we included the code presented in 1.3.4 into our solver, it's necessary to add it too.
We can now take a look at the content of `thermophysicalProperties`:

```
thermoType
}
    type                heRhoThermo;
    mixture             pureMixture;
    transport           const;
    thermo              hConst;
    equationOfState     rhoConst;
    specie              specie;
    energy              sensibleInternalEnergy;
}

mixture
{
    specie
    {
        molWeight       23;
    }

    equationOfState
    {
        rho             923.7;
    }

    thermodynamics
    {
        Cp              1.3796e3;
        Hf              1.131e5;
    }

    transport
    {
        mu              6.5442e-4;
        Pr              0.0102; // k = 88.747 W/(m*K)
    }
}
```

All keywords under the section `thermoType` specifies which physical models will be

used to calculate our quantities, then all necessary numerical parameters will be set in the corresponding section inside `mixture`. All choices will be rapidly explained:

- `type heRhoThermo` specifies that our thermophysical model is based on ρ.
- `mixture pureMixture` specifies that our fluid is composed of only one substance.
- `transport const` set μ and *thermal conductivity* k_c ($\mathrm{W\,m^{-1}\,K^{-1}}$) as *constants*. Below, in the corresponding `mixture` section, all values are declared. For some reasons, OpenFOAM requires to assign a value to the *Prandtl number* `Pr`, defined as:
$$Pr = \frac{c_p \mu}{k_c} \tag{1.27}$$
It is important to notice that also c_p must be declared as constant and reported into the code before `Pr`. For the sake of completeness, we commented in the code the value of k_c.
- `thermo hConst` assumes constant c_p, which value is specified below, together with *heat of fusion* ($\mathrm{J\,kg^{-1}}$) `Hf`.
- `equationOFState rhoConst` declares that our equation of state will be:
$$\rho = const \tag{1.28}$$
It's value is then specified below.
- `specie specie` allows to specify number of moles and molar weight for every component of the mixture. Since we will deal wth liquid sodium, its atomic weight is reported.
- `energy sensibleInternalEnergy` specifies that e will be the variable of the energy equation presented in 1.3.3. In fact, one can choose to use *specific hentalpy* ($\mathrm{J\,kg^{-1}}$) instead.

It is now very important to explain why all quantities, ρ included, are taken as constants: they all depends mainly on temperature in case of sodium. Since we are considering no thermal effects, $T = const$ and so all quantities depending on it will be constant too. Moreover, $\rho = const$ is the state equation characterising incompressible fluids. In order to get the reported values we used the following empirical correlations:

- $\rho = 1014 - 0.235T$
- $c_p = 1608 - 0.7481T + 3.929 * 10^{-4}T^2$
- $\mu = exp(\frac{556.835}{T} - 0.3958\ln(T) - 6.4406)$
- $k_c = 110 - 0.0648T + 1.16 * 10^{-5}T^2$

in which $T = 380(\mathrm{K})$ has been assumed. They were all taken from [Sobolev, 2010].

constant/transportProperties

Here numerical values are assigned to the fluid properties related to the interaction with magnetic fields, together with their unit of measure:

```
muB             [1 1 0 0 0 -2 0] 1.257e-6;

sigma           [-1 -3 1 0 0 2 0] 1.011e7;
```

While the value of μ_B can be considered independent from temperature with good accuracy, σ is considered constant only because we are not dealing with thermal effects. The value is taken from the following correlation, considering $T = 380(\mathrm{K})$:

$$\eta = (3.126 + 6.218 * 10^{-3}T + 3.093 * 10^{-5}T^2) * 10^{-8} \tag{1.29}$$

and remembering that $\sigma = 1/\eta$.

constant/turbulenceProperties

Here it's possible to set a turbulence model for the simulation. Since we are dealing with laminar flows, no turbulence model is necessary, decision specified by this only entry:

```
simulationType laminar;
```

system/blockMeshDict

The remaining directory, `system`, is dedicated to all simulation settings, like mesh definition, time step management, numerical schemes and so on.
The first entry is `blockMeshDict`, where mesh is defined under every aspect, starting from vertexes coordinates definition:

```
convertToMeters 0.01;

vertices
(
    (0 -1 -0.005)
    (20 -1 -0.005)
    (20 1 -0.005)
    (0 1 -0.005)
    (0 -1 0.005)
    (20 -1 0.005)
    (20 1 0.005)
    (0 1 0.005)
);
```

It is important to notice that, even though our simulation is 2D, OpenFOAM allows only the creation of 3D meshes, so our rectangle is initially defined as a very thin parallelepiped. The `convertToMeters` keyword allows us to define the physical dimensions of our domain. We will explain how with an example: the second vertex

is positioned $20 * 0.01 = 0.2$(m) from the origin along x, $-1 * 0.01 = -0.01$(m) along y and $-0.005 * 0.01 = -0.00005$(m) along z. This implies that our mesh is 20(cm) in length, 2(cm) in height and 0.01(cm) in thickness. Lastly, it is important to mention the fact that OpenFOAM enumerates all vertexes with a label, starting from 0 (first entry) to 7 in our case (last entry). As we wil see, these labels will be used later.
The second piece of code takes advantage of the aforementioned labels to define all blocks composing the mesh:

```
blocks
(
    hex (0 1 2 3 4 5 6 7) (400 60 1) simpleGrading (
                                                       1
                                                       (
                                                       (0.5 0.5 4)
                                                       (0.5 0.5 0.25)
                                                       )
                                                       1
                                                   )
);
```

Our mesh is thus composed of only one hexaedron (`hex`), divided in 400 elements along x, 60 along y and only 1 along z, another important feature that characterizes a 2D simulation[22]. The keyword `simpleGrading` finally allows us to define a *grading* for the mesh elements size. Without entering too much into details, the reported code reduces their dimension along y when moving closer to walls. Figure 1.4 shows a mesh section.

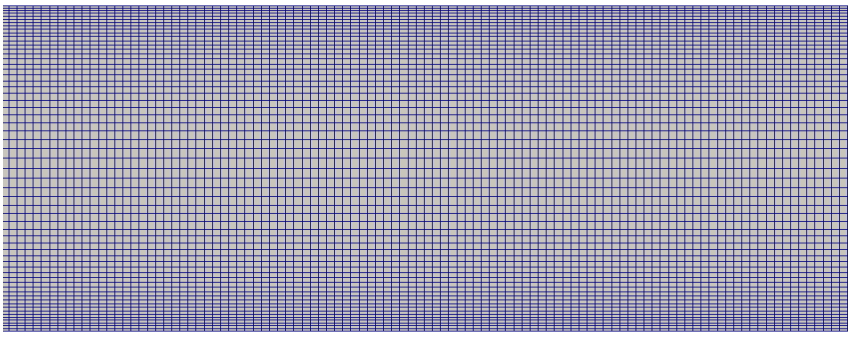


Figure 1.4. *Section of the mesh, showing the grading of elements close to walls.*

The reason behind our mesh arrangement has to be searched inside the physics of the Hartmann flow: when moving towards walls, the analytical solution (1.22) shows a steep gradient till 0, defining the so-called *Hartmann layer*, whose thickness δ is given by:

$$\delta = \frac{1}{B_0}\sqrt{\frac{\mu}{\sigma}} = \frac{L}{Ha} \tag{1.30}$$

[22]To summarize, a 2D simulation requires the definition of only one element along one of the three direction, plus the declaration of faces perpendicular to that same direction as empty.

so the higher the Ha, the thinner the δ. Since we will use our mesh for a wide range of Ha values, it is better to have a good grading from the start.
Lastly, The face declaration is next:

```
boundary
(
    inlet
    {
        type patch;
        faces
        (
            (0 4 7 3)
        );
    }
    outlet
    {
        type patch;
        faces
        (
            (2 6 5 1)
        );
    }
    lowerWall
    {
        type patch;
        faces
        (
            (1 5 4 0)
        );
    }
    upperWall
    {
        type patch;
        faces
        (
            (3 7 6 2)
        );
    }
    frontAndBack
    {
        type empty;
        faces
        (
            (0 3 2 1) (4 5 6 7)
        );
    }
);
```

In each declaration, every face receives: a name, a type (`patch` for generic boundaries, `empty` for empty faces) and at least one quadruplet of numbers, representing the four delimiting vertexes[23].

system/controlDict

This file is essentially a list of keywords, the majority of which specifies time settings. The most important are reported and explained.

- `application` allows to specify which OpenFOAM solver will be used. Our entry will be `comprMhdFoam`, the solver we presented in 1.3.

- `endTime` determines when the solver will stop advancing our equations in time[24]. Here there is no fixed entry for us , since the minimum amount of time which allows our flow to fully develop is inversely proportional to Ha. Indeed the stronger $\vec{B}$ is, the lower the laminarization time will be.

- `deltaT` sets the time step. It should be chosen so that the following is satisfied:

$$Co = \frac{u_x \Delta t}{\Delta x} + \frac{u_y \Delta t}{\Delta y} + \frac{u_z \Delta t}{\Delta z} \leq 1 \tag{1.31}$$

  where Δt is clearly the time step and $\Delta x, \Delta y, \Delta z$ are the minimum spatial interval inside the mesh, respectively for the three directions. Co is called *Courant number*, while (1.31) is the *Courant–Friedrichs–Lewy condition*, usually abbreviated as *CFL condition*. The Courant number provides a comparison between the space that information covers in Δt under the influence of velocity field and the spatial discretization, so if it was bigger than 1, some information would be lost. Moreover, if CFL is not respected, numerical instabilities will usually arise.

- `writeControl` allows us to choose which time controls the writing of results: it can be real time, CPU time, simulated time or a certain number of time steps. We opted for simulated time, which correspond to the `runTime` entry.

- `writeInterval` specifies the time interval between the writing of results. Since we previously chose `runTime` as entry for `writeControl`, we will deal with simulated time intervals. Also in this case there is no fixed value preferable to others: if one has great storage capability, a small value can make data sampling very accurate[25]. Here we opted for numbers which will result roughly in 20 to 50 data savings.

- `maxCo` allows us to set a maximum value for the Courant number. It's a very useful tool, since the entry in `deltaT` can be overwritten in order to satisfy

[23] Vertexes order is defined by watching the face from inside the block and by listing labels clockwise, starting from any of them.

[24] It is also possible to set a starting time, but it will always be 0 in our case.

[25] If simulated time is chosen, the ratio between the total time and the aforementioned interval of time represents the number of times results are saved, which gives an idea of how much memory will be necessary.

this restrain. To make sure our simulation will be stable, we imposed 0.5 as maximum Co, so that also (1.31) is generously satisfied.

- `adjustTimeStep` must be set to `on` in order to allow time step modification.

system/decomposeParDict

Here one can specify setting for *parallel computing*, which consists in splitting the domain in a given number of parts, then assign a processor core to each one of them to make calculation. All cores then communicate with each other in order to correctly advance the solution. Its content is the following:

```
numberOfSubdomains  4;

method     scotch;
```

The first keyword specifies the number of subdomains (we will use a quad core processor), while the second sets the criterion to follow in the decomposition. Our entry `scotch` requires no geometric inputs from the user and attempts instead to minimize the number of core boundaries, i. e., the need of cores to communicate with each other. Parallel computing is a great tool to speed up a simulation, so it is usually recommended[26].

system/fvSchemes

Here are specified discretization schemes for all differential operators. We didn't go deeply into this topic and let almost everything at their default settings. Only two entries were addressed: time derivatives and the advection term $(\vec{u} \cdot \nabla)\vec{u}$, which appears in the momentum equation (1.16). Firstly, time derivatives:

```
ddtSchemes
{
    default         backward; //Euler;
}
```

With respect to the previous one (left commented), our entry is a second order scheme, ensuring more accuracy while advancing in time. Obviously it is more demanding in terms of computational cost.
Regarding the advection term, our new entry is the following:

```
divSchemes
{
    div(phi,U)      Gauss linearUpwind grad(U); //upwind;
}
```

What we discussed regarding time derivatives holds also in this case: better scheme at a higher cost.

[26] In case of meshes with a large amount of elements, a high number of cores is obviously very useful. On the contrary, splitting a low elements mesh in too many parts can result in the opposite effect, due to the high time needed for communications between cores.

system/fvSolution

Here the numerical schemes adopted to solve all equation are reported, together with settings related to PIMPLE and BPISO cycles (the second is the one controlling how many times (1.14) is solved). Firstly, it is important to report the schemes of p and $\vec{B}$ equations, since they differ from default entries:

```
solvers
{
    p
    {
        solver              GAMG;
        smoother            GaussSeidel;
        maxIter             100;
        nCellsCoarsestLevel 155;
        tolerance           1e-06;
        relTol              0;
    }

    B
    {
        solver              GAMG;
        smoother            GaussSeidel;
        maxIter             100;
        nCellsCoarsestLevel 155;
        tolerance           1e-07;
        relTol              0;
    }
}
```

For both, the `GAMG` (generalised geometric-algebraic multi-grid) solver, coupled with the `GaussSeidel` smoother, is the optimal choice, since it provides convergence to the specified tolerance in lesser iteration with respect to other options. To make good use of it, it's better to set `nCellsCoarsestLevel` to a value similar to the square root of the number of mesh elements[27]. Moreover, it's usually a good idea to set `maxIter` to a relatively small value, otherwise it could greatly reduce calculation speed, especially in the early stage of the simulation.
All PIMPLE options are now shown:

```
PIMPLE
{
    momentumPredictor        yes;
    transonic                no;
    nOuterCorrectors         10;
    nCorrectors              2;
```

[27]GAMG uses the principle of: generating a quick solution on a mesh with a small number of cells, mapping this solution onto a finer mesh, using it as an initial guess to obtain an accurate solution on the fine mesh.

```
    nNonOrthogonalCorrectors 0;
    consistent               yes;
    simpleRho                yes;

    pMaxFactor               1.5;
    pMinFactor               0.9;

    outerCorrectorResidualControl
    {
        p
        {
            relTol           0;
            tolerance        0.01;
        }

        e
        {
            relTol           0;
            tolerance        0.005;
        }

        "(U|k|epsilon)"
        {
            relTol           0;
            tolerance        0.005;
        }
    }

    turbOnFinalIterOnly no;
}
```

The most noticeable entries for our case are the following:

- `nOuterCorrectors` sets the maximum number of PIMPLE iterations. It should be high enough to ensure convergence after a little amount of simulated time, but not excessively big, otherwise it would slow calculation down[28].
- `nCorrectors` decides how many times pressure equation is solved inside one PIMPLE iteration. It should be set at least to 2, since pressure tends to be the bottle neck for general cycle convergence.
- `outerCorrectorResidualControl` allows to decide tolerances for PIMPLE convergence, which happens only when a cycle starts with all fields residuals lower than the corresponding `tolerance` value. For precise results, 0.005 is small enough. The reason why we used 0.01 for p is that we are not interested in

[28] In the very early stages of a simulations, until boundary layers are not properly developed, convergence is naturally really hard and it's not wise to force it.

pressure-related phenomena and our choice speeds calculation up considerably. k and ϵ are specified too, in case one wants to use RAS turbulence models.

The last entries are *relaxation factors*, briefly expained in footnote 15 inside 1.3.2:

```
relaxationFactors
{
    fields
    {
        p               0.3;
        rho             1;
    }
    equations
    {
        U               0.9;
        e               0.5;
        "(k|epsilon|omega)" 0.9;
    }
}
```

All entries are pretty standard, except for p, which value is lower than usual to help convergence, and ρ, which factor isn't even used by the solver since its value is set to constant inside `thermophysicalProperties`. As for k and ϵ, also ω is a quantity introduced by using RAS models.

system/singleGraph-x

This last file allows us to sample data in a plot-friendly format, basically gathering fields values along a line placed inside the mesh for every instant of time in which data are saved. Its content is the following:

```
start   (x -0.01 0);
end     (x 0.01 0);
fields  (U);

#includeEtc "caseDicts/postProcessing/graphs/sampleDict.cfg"

setConfig
{
    type  lineUniform;
    axis  y;
    nPoints 100;
}

#includeEtc "caseDicts/postProcessing/graphs/graph.cfg"
```

`start` and `end` sets the limits of the line. `x` entry refers to a generic position along the corresponding axis, while y and z are fixed so that our line will be parallel to the y direction. `fields` specifies which fields values will be saved. Under `setConfig` one

can instead define how many sampling points will be present and which distribution criterion they will follow along our line, while remarking also its parallel axis.
It is important to mention that one can decide to place multiple copies of this file inside the `system` folder. Since we wanted to monitor data in various sections of our geometry, we decided to place sampling lines every 3(cm), thus obtaining 6 files in total, each one with a different entry for `x`: 0.03, 0.06, 0.09, 0.12, 0.15, 0.18. Finally, in order to properly save all results, it is essential to insert the following portion of code at the end of `controlDict`:

```
functions
{
    #includeFunc singleGraph-0.03
    #includeFunc singleGraph-0.06
    #includeFunc singleGraph-0.09
    #includeFunc singleGraph-0.12
    #includeFunc singleGraph-0.15
    #includeFunc singleGraph-0.18
}
```

1.4.3 Results

Simulations were run for various Ha values: 1, 5, 20, and 50. Results were then imported on Matlab, thanks to a custom-made function, which is able to read data gathered by using the `singleGraph` sampling method. Basically, the function, called `readFields`, requires the user to specify which section he's intersted in (in our case one of the six specified above), then browse all folders containing the text files where data are printed, read them and saves them in matrices, one for each quantity of interest (in case of $\vec{u}$, one for each of its components). These are organized so that every column represents the field of interest at a given instant of time, therefore, by plotting column values in sequence, one can see the evolution of that field along a given section of the geometry.
In order to check the goodness of our solutions with respect to the analytical profile for every one of the aforementioned Ha values, we placed ourselves at the minimum distance from the outlet that allowed us to see the best-fitting result. Firstly, the relative error integrated along the section is reported for every Ha value in Table 1.1, followed by Figure 1.5, where a direct comparison between analytical and OpenFOAM profiles is shown.

Hartmann number	error (%)
1	0.0848
5	0.0720
20	0.0555
50	0.0740

Table 1.1. *Relative error integrated along the chosen section for every tested Hartmann value.*

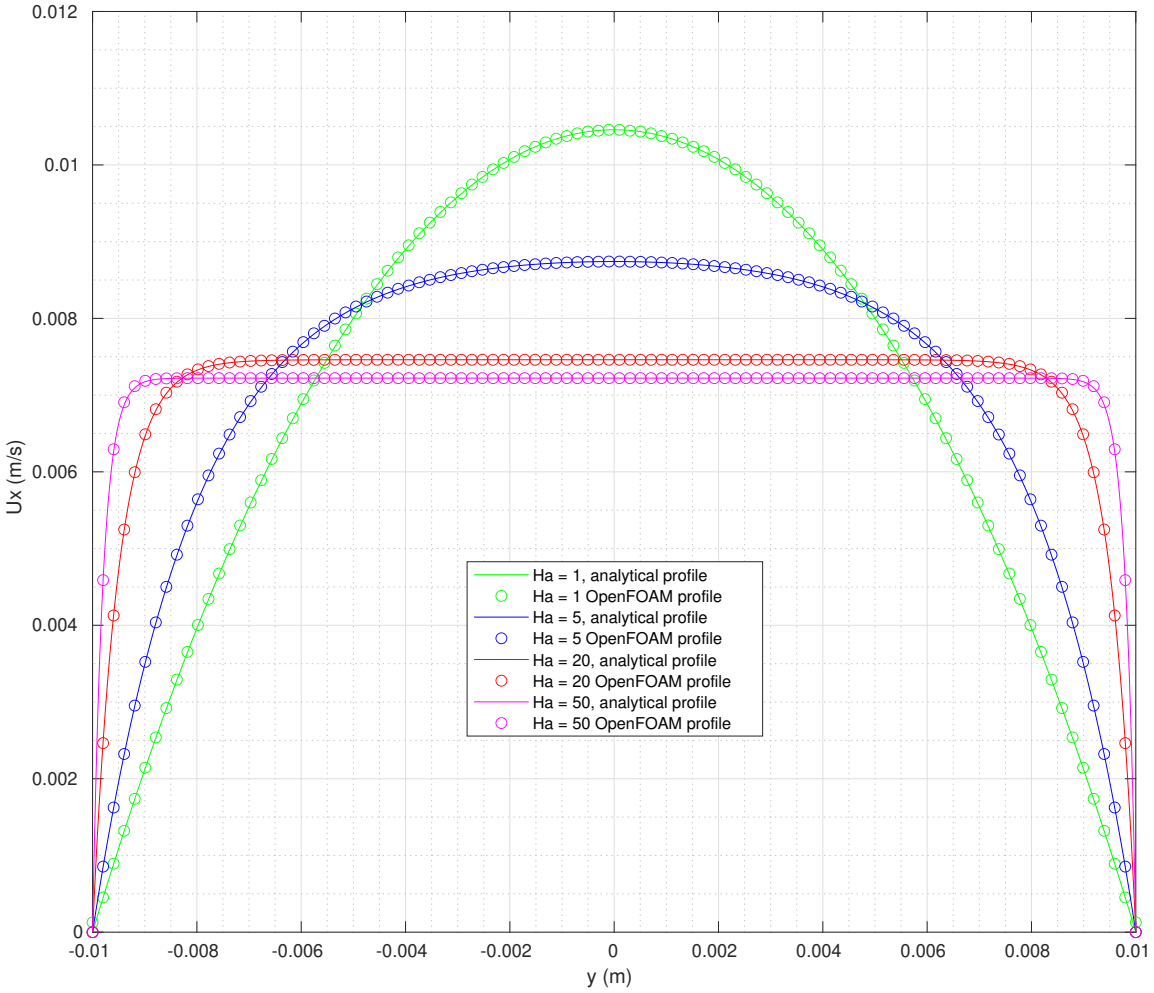


Figure 1.5. *Comparison between analytical profiles (continuous lines) and OpenFOAM results (dots).*

Both numbers and plots confirm that the verification has been successfull. In particular, the error never goes beyond 0.1%, which is a proof of very good accuracy.

1.5 Conclusions

In this first chapter, the implementation of the new solver comprMhdFoam has been presented, providing physical motivations to every choice, and successfully verified on the Hartmann flow case. It is important to stress the fact that the employed case was chosen in order to test only the goodness of the solver in reproducing MHD effects. Indeed, as already mentioned, its capability of treating compressibility phenomena is assured, given the good reputation of rhoPimpleFoam.
The next step will be the application of turbulence models in the presence of magnetic effects, so that to test their reliability in such conditions. It will be also a good occasion to study a more complex MHD system with respect to the Hartmann flow case.

Chapter 2

MHD flow across a backward facing step

In the second chapter, the capability of the new solver to efficiently employ turbulence models in the study of MHD systems will be verified. For such purpose, we will take into consideration the flow of a conductive fluid across a backward facing step under the effect of an external magnetic field perpendicular to the direction of motion. At first, a mesh sensitivity analysis without any turbulence model will be operated in order to show why they are extremely useful in such scenarios. After that, Reynolds Averaged Simulation (RAS) and Large Eddy Simulation (LES) will be introduced. For both, we will report a theoretical background before proceeding with the associated results.

2.1 Introduction

After the verification of the new solver comprMhdFoam, the next step is to test its goodness in employing turbulence models for the study of MHD systems which requires them. Briefly, *turbulence modeling* is the construction and use of a mathematical model to predict the effects of turbulence on a given system. In spite of decades of research, there is no analytical theory to predict the evolution of turbulent flows, so they are developed in order to provide simplified constitutive equations which try to predict the statistical evolution of turbulent flows. We decided to give priority to such topic over compressibility effects because we are confident in the fact that the latter are not a problem for the new solver, even in presence of a magnetic field, while there is more uncertainty regarding the application of turbulence models. Another motivation is the scarcity of works in the literature pertaining to the employment of turbulence models in MHD cases, particularly when speaking about engineering applications.
The physical system we chose to study the proposed topic is the flow across a *backward facing step*. It consists in a channel created by two infinitely extended parallel plates, with one of the two undergoing a sudden step-like enlargement. Figure 2.1 shows a section perpendicular to walls and anticipates an idea of the fluid motion.

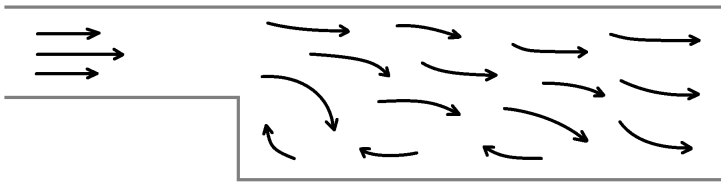


Figure 2.1. *Section of the backward facing step taken perpendicularly to walls. Arrows show an idea of the fluid motion.*

The backward facing step scenario has been chosen due to its simple-to-reproduce geometry and to the large quantity of studies related to it ([Tano-Retamales et al., 2019] is an example). Indeed, it is well known that, for a range of Reynolds number values, the step will induce the fluid to create a recirculating region, followed by a point in which the motion reattaches to the wall. The streamline plot in Figure 2.2 is useful to visualize the phenomenon.

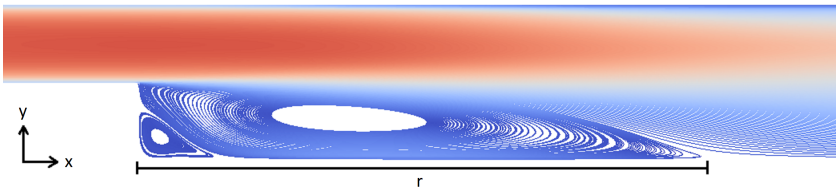
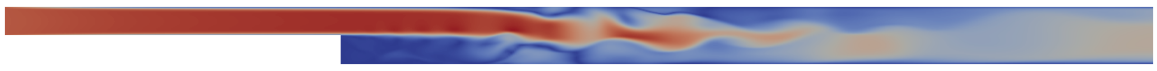


Figure 2.2. *Streamlines plot showing recirculating region. Colors show the magnitude of velocity field, increasing from blue to red.*

The length indicated as r is called *reattachment length* (m). When the fluid motion stabilize in such configuration, studies show that a two-dimensional simulations is accurate enough to provide a sufficiently precise solution, given the plane symmetry of the geometry ([Tano-Retamales et al., 2019] uses this approach).
Over a certain Re value, the motion becomes more chaotic and the recirculating region cannot be clearly defined. This situation is ideal to show the effect of an external magnetic field: as explained by [Trotta, 2019], if the fluid is conductive, a magnetic field perpendicular to the direction of motion (and also perpendicular to walls in this case) has stabilizing properties on the flow, in the sense that it obstructs the development of turbulence, as illustrated in Figure 2.3).



(a) $Ha = 0$

(b) $Ha = 40$

(c) $Ha = 100$

Figure 2.3. *Comparison between three cases with fixed Re and different Ha.*

In the three cases reported in Figure 2.3, $Re = 10000$ and Ha varies between 0, 40 and 100. While it's impossible to define r in (a), given the turbulent motion, it's evident in (b) and (c).
In order to test the goodness of the new solver in employing turbulence models to study MHD scenarios, the aforementioned effect will be addressed. We will start with a *mesh sensitivity* analysis, without adopting any model. The results will show the utility of such tools. *Reynolds Averaged Simulation* (RAS) and *Large Eddy Simulation* (LES) turbulence models will then be introduced and employed to study the phenomenon. Their goodness will be verified through a comparison with the results of the mesh sensitivity analysis.

2.2 Mesh sensitivity analysis

The mesh sensitivity analysis will be carried out approximating the problem as two-dimensional, given the fact that it's a good approach. Moreover, simulations will be significantly faster[1]. They will all be performed using comprMhdFoam, while the post process will be executed on Matlab. For the OpenFOAM case setup, we can refer to Section 1.4.2, since the folder tree and most of the files are exactly the same. Every difference will be pointed out and analyzed afterwards.

2.2.1 Mesh definition

The geometry will now be presented by taking advantage of Figure 2.4.

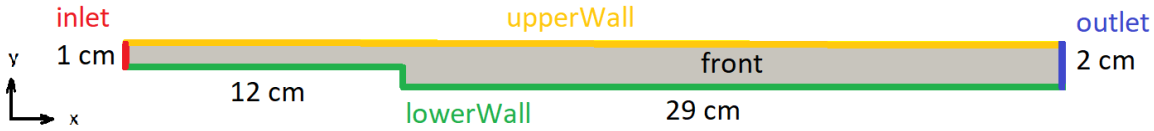


Figure 2.4. *Geometry of our case, provided with dimensions and boundary names.*

Dimensions are taken from [Tano-Retamales et al., 2019], since they obtained precise results by employing turbulence models in OpenFOAM, LES in particular, adopting the same geometry[2]. In order to be clearer, we will keep the names employed in Section 1.4.2. This implies that the `blockMeshDict` file will be very similar to the already presented one in structure, besides for the number of vertexes and their order and for blocks declaration and their grading. Regarding this last feature, we cannot resort to mesh refinement only near walls, but rather across the entirety of the step region, since turbulence will develop for all the height of our channel. For this reason the grading will only be oriented in the x direction, so that elements far from the step, where turbulent phenomena are less important, will be a bit longer, thus reducing the total number of cells.

```
convertToMeters 0.001;
```

[1] Indeed, a 2D mesh contains way less cells than a 3D one with the same level of refinement.
[2] Because of their good results, it's safe to assume that these dimensions are sufficient to ensure proper turbulence development, while limiting the amount of elements composing the mesh.

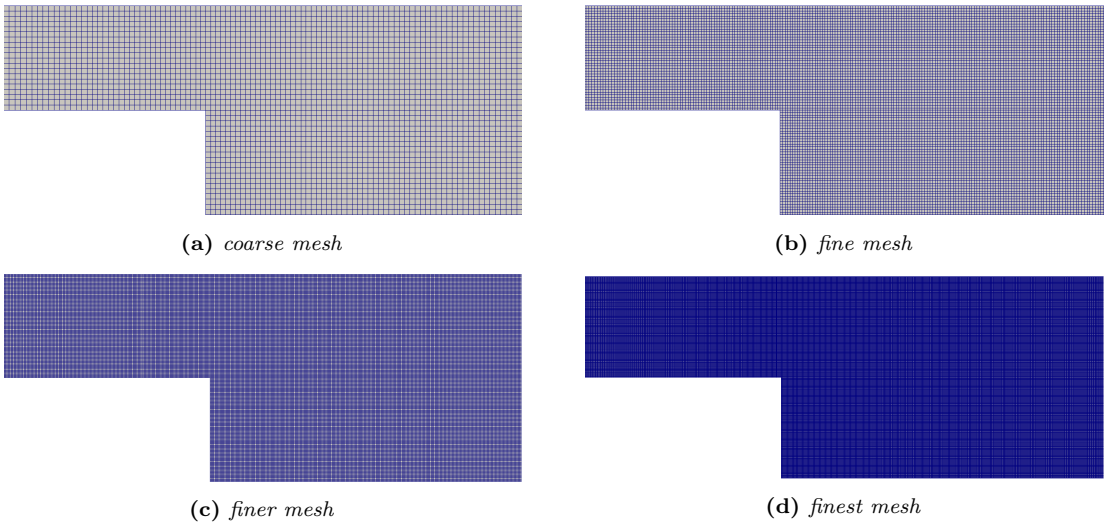
```
vertices
(
    (-120 0 -0.1)
    (0 0 -0.1)
    (0 -10 -0.1)
    (290 -10 -0.1)
    (290 0 -0.1)
    (290 10 -0.1)
    (0 10 -0.1)
    (-120 10 -0.1)
    (-120 0 0.1)
    (0 0 0.1)
    (0 -10 0.1)
    (290 -10 0.1)
    (290 0 0.1)
    (290 10 0.1)
    (0 10 0.1)
    (-120 10 0.1)
);

blocks
(
    hex (0 1 6 7 8 9 14 15) (175 20 1) simpleGrading (0.5 1 1)
    hex (2 3 4 1 10 11 12 9) (423 20 1) simpleGrading (2 1 1)
    hex (1 4 5 6 9 12 13 14) (423 20 1) simpleGrading (2 1 1)
);
```

As we can see, the declaration of three different hexaedra is necessary to compose the geometry reported in Figure 2.4. The first one represents the inlet block, while the other two the outlet one. When creating meshes made of more than one block, it's important that matching faces contains the same number of elements because it ensures a more precise calculation.
Gradings are set so that the cells will shorten starting from the inlet and moving along x direction, then will start to enlarge once the step is surpassed.
The reported code refers to the coarsest of our meshes. A comparison between the four levels of refinement we will adopt is reported in Table 2.1, in terms of: number of cells, minimum and maximum cells Δx, cells Δy (grading is only along x) and maximum aspect ratio (maximum $\Delta x/\Delta y$). A visual comparison is reported afterward in Figure 2.5.

Mesh	cells	Δx_{max} (m)	Δx_{min} (m)	Δy (m)	max aspect ratio
Coarse	20420	$9.50385 * 10^{-4}$	$4.75163 * 10^{-4}$	$5 * 10^{-4}$	1.90077
Fine	81680	$4.75247 * 10^{-4}$	$2.37592 * 10^{-4}$	$2.5 * 10^{-4}$	1.90099
Finer	326720	$2.37637 * 10^{-4}$	$1.18799 * 10^{-4}$	$1.25 * 10^{-4}$	1.9011
Finest	1306880	$1.18822 * 10^{-4}$	$0.594002 * 10^{-4}$	$0.625 * 10^{-4}$	1.90115

Table 2.1. *Comparison between parameters pertaining to the four meshes.*



(a) *coarse mesh* **(b)** *fine mesh*

(c) *finer mesh* **(d)** *finest mesh*

Figure 2.5. *Step section of the four meshes.*

2.2.2 Other case setup differences

All the other worth-noting differences with respect to 1.4.2 will be now adressed. The first one is in the velocity inlet:

```
inlet
    {
        type            fixedValue;
        value           uniform (0.71 0 0);
    }
```

Since the current value is a hundred times bigger that the one adopted in 1.4.2, we can expect $Re = 10000$, a value that ensures a turbulent motion[3], if not obstructed by $\vec{B}$. Regarding the just mentioned field, its value on the boundaries will still be set in order to guarantee the desired Ha value.
The second difference consists in setting now the maximum value of the Courant number to a higher one: after some testings, we decided that 0.9 is low enough to ensure convergence.
One last mention goes to the number of subdomains we will use: in order to speed up the calculation, we employed a 15-core processor to carry out simulations on all of our meshes, except for the coarsest one, where such high number of cores would cause a slowdown instead.

2.2.3 Results

In the mesh sensitivity analysis, the aim is to determine how much time the magnetic field will employ to fully laminarize the flow, i. e., the *laminarization time* (s). What we expect is that, by augmenting the mesh refinement, such time interval will increase.

[3]All other quantities inside (1.24) are the same, since we will still consider liquid sodium at $T = 380$(K) as our fluid and the half-length of the channel after the step as L.

$Ha = 24$[4] will be adopted, since it ensures turbulence suppression at the point where we can detect differences between our four meshes. Indeed, in case of a too high value, we wouldn't notice many differences, since laminarization would occur in a very short time. Regarding time features, we will set 8 seconds as limit, estimating simulations on the finest mesh to be considerably long[5].
A simple Matlab script will be employed to estimate the precise laminarization time. After reading the velocity profiles at a given section, it goes through all time instants t_i and performs the following check on them:

$$\frac{\max\{u(y,t_i) - u(y,t_{i-1})\}}{u_{avg}(t_{i-1})} \leq 10^{-3} \wedge \frac{\max\{u(y,t_{i+1}) - u(y,t_i)\}}{u_{avg}(t_{i-1})} \leq 10^{-3} \tag{2.1}$$

where u is the magnitude of $\vec{u}$, which depends spatially only on y (we are considering a given section, so x is fixed) and u_{avg} represents its average value across the same section. We check on two consecutive time instants because it can happen that t_i passed the test, while t_{i+1} wouldn't. A visual check on profiles plot was sufficient to confirm the accuracy of (2.1)[6].
Figure 2.6 shows the aforementioned profiles at different time steps. They are taken 6 cm away from the step[7].

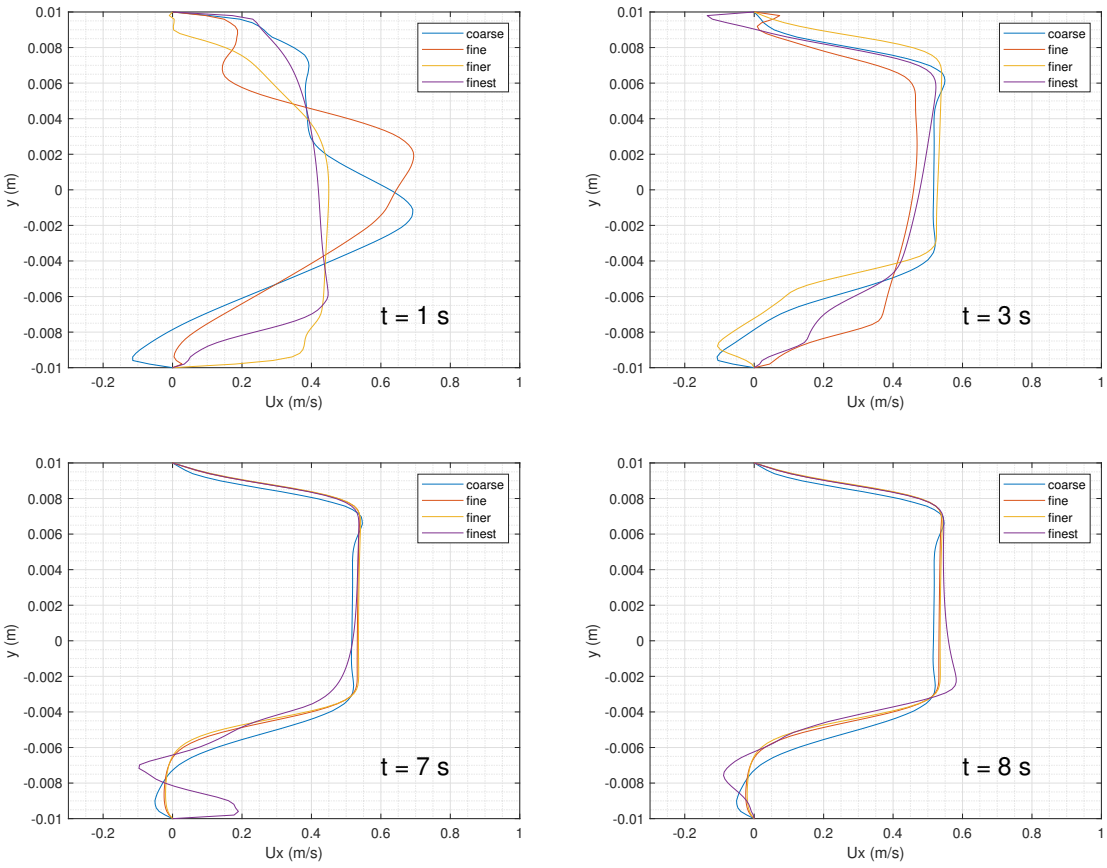


Figure 2.6. *Comparison between u_x profiles taken* 6 cm *away from the step for the different meshes.*

[4]We considered L as the half-length of the channel after the step.
[5]Such simulation lasted for about 8 days on a 15-core processor.
[6]The visual check was also necessary to establish 10^{-3} as tolerance.
[7]The steady state shape is motivated by the presence of the recirculating region.

All laminarization times are calculated at 27 cm from the step. They are gathered in Table 2.2:

Mesh	Laminarization time (s)
Coarse	4.8
Fine	6.6
Finer	6.4
Finest	Not in time limit

Table 2.2. *Comparison between laminarization time for the four meshes.*

As expected, the coarse mesh shows the most stable behavior, the two in the middle have a bit longer laminarization time, while remaining similar, and the finest one doesn't fully laminarize in the time limit. This can be explained considering the effect of numerical approximation on turbulence: in order to discretize the geometry, OpenFOAM employs the *Finite Volume Method*, which divides the domain in cells (following the instructions inside blockMeshDict) and solves a discretized version of every equations in each one of them, saving only one value for every field inside every cell center. This greatly affects how turbulence is simulated: from theory and facts we know that eddies are born with a characteristic dimension which characterizes the geometry, called *macroscale*, where energy is supplied. Then they start to decay in smaller eddies, starting the so-called *energy cascade*, until they reach a point in which viscous effects dissipate all energy. The dimension they have at that point is called *microscale*. Kolmogorov in his famous theory of 1941 stated the following:

$$\frac{\lambda}{\Lambda} \sim Re^{-\frac{3}{4}} \tag{2.2}$$

where λ is the microscale and Λ is the macroscale, which corresponds in our case to the half length of our channel after the step. If we consider Re to be around 10000, for us $\lambda \sim 10^{-5}$(m). Now, if we consider that cells close to the step in the coarse mesh have a dimension in the order of $0.02/40 = 5 * 10^{-4}$(m), we can understand why it's the more stable situation: a lot of turbulence scales get lost inside cells. This phenomenon is called *numerical viscosity*, since it has the same effect of fluid viscosity, but at the scale of mesh elements. This explains the results reported in table 2.2: the more the mesh is refined, the more accurate the result.
At this point it seems that, if one wants to properly simulate turbulence, the only way is to refine the mesh to the point in which elements are smaller (or at least similar) to the microscale[8], which would result in drastically long simulations. Fortunately, there is a way around: the employment of *turbulence models*, which allows to simulate the presence of turbulent phenomena without the need to refine the mesh to extreme levels.

2.3 Reynolds Averaged Simulations

The first model employed will be the *Reynolds Averaged Simulation*, which consists in solving a time-averaged version of hydrodynamics equations presented in 1.2.1.

[8]This approach goes by the name *Direct Numerical Simulation* (DNS).

2.3.1 Theoretical background

The idea behind RAS is the so-called *Reynolds decomposition*, which implies that every field will be written as sum of a part constant in time (indicated with $\overline{\bullet}$) and a fluctuating one (indicated with $\bullet'$):

$$f(\vec{x},t) = \overline{f(\vec{x})} + f'(\vec{x},t) \tag{2.3}$$

Fluctuating terms benefit from the following property:

$$\overline{f'} = 0 \tag{2.4}$$

By means of (2.3) and (2.4)[9] one can get to a set of equations only for averaged quantities, except for one very important term:

$$\overline{\overline{\tau}}_R \equiv -\overline{\rho\vec{u}' \otimes \vec{u}'} \tag{2.5}$$

$\overline{\overline{\tau}}_R$ is called *Reynolds stress tensor* (Pa). Together with α_{eff} in (1.21), it contains information about the turbulent behavior of the flow, since it depends on velocity fluctuations[10]. In order to close our system, it should be properly modeled. A common way is to consider the following:

$$\overline{\overline{\tau}}_R = -\frac{2}{3}\overline{\rho}\kappa\overline{\overline{I}} + \mu_T\left[(\nabla\vec{u} + \nabla\vec{u}^T) - \frac{2}{3}(\nabla\cdot\vec{u})\overline{\overline{I}}\right] \tag{2.6}$$

where $\kappa = \frac{1}{2}\overline{\vec{u}'\cdot\vec{u}'}$ is called *turbulent kinetic energy* ($\mathrm{m^2\,s^{-2}}$) and μ_T is the *turbulent dynamic viscosity* (Pa s). The modeling of this last quantity determines how the system will be closed: one common way are the *two equations* models, which consider μ_T as function of two quantities, described by their own equations. We will adopt this last approach, in particular considering the so-called $\kappa-\omega$ model, where:

$$\nu_T \equiv \frac{\mu_T}{\overline{\rho}} = \frac{\kappa}{\omega} \tag{2.7}$$

ν_T is called *turbulent kinematic viscosity* ($\mathrm{m^2\,s^{-1}}$). Authors interpreted ω ($\mathrm{s^{-1}}$) in many ways: for someone it was the root mean square of vorticity fluctuations[11], for others the ratio between the dissipation rate of turbulent kinetic energy and the turbulent kinetic energy itself. The equation for κ can be obtained by subtracting the momentum equation for averaged quantities to the complete one and then multiplying all for $\vec{u}'$. At this point, the equation for ω can be totally invented: usually it can be identical to the one for κ, except for some coefficients, which are set by means of benchmark simulations.
Before proceeding with our results, it is important to mention a tool usually adopted in RAS to further reduce computational cost, which are *wall functions*. They are empirical laws which explains how the flow will behave when approaching a wall and for this reason they are implemented as boundary conditions in OpenFOAM. Their advantage lies in the fact that, by employing them, one doesn't need to refine the mesh close to boundaries in order to obtain acceptable results[12].

[9] After having rewritten all fields, one must average equations in time in order to take advantage of (2.4). Indeed all terms like $\overline{\overline{f}g'}$ will be equal to 0.

[10] Since the averaging of (1.4) is a long process, it's beyond our purpose to report all terms containing fluctuations. We will limit ourselves to report the way OpenFOAM models turbulence inside the averaged (1.4), which is the assignment of a proper value to α_{eff}.

[11] Vorticity is defined as $\omega \equiv \nabla\times\vec{u}$.

[12] A good refinement brings always more correct results, but is usually very expensive.

2.3.2 Results

A comparison between RAS and no-model profiles will be reported in Figure 2.7 and laminarization times in Table 2.3, employing only the coarse mesh as first verification. $Ha = 24$ is adopted to give continuity with respect to results of Subsection 2.2.3.

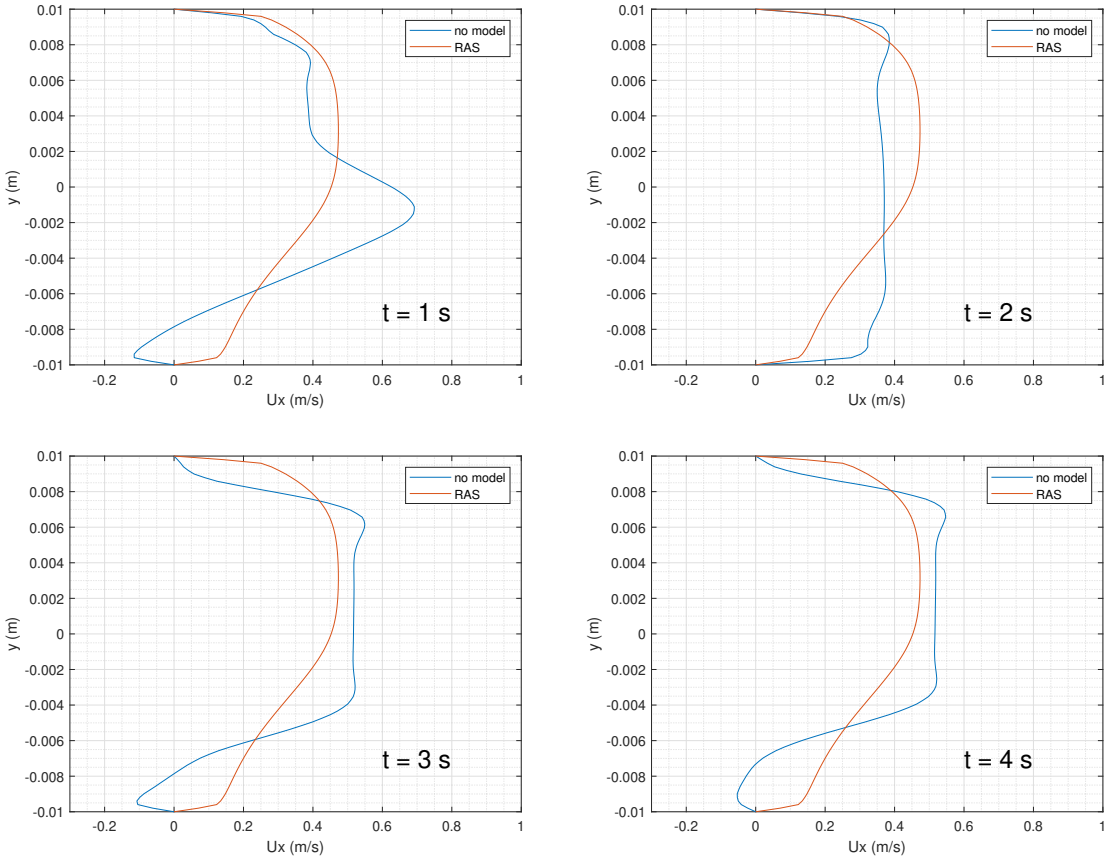


Figure 2.7. *Comparison between u_x profile without model and with RAS model.*

Model	Laminarization time (s)
No model	4.8
RAS	1.4

Table 2.3. *Comparison of laminarization time without model and with RAS model.*

Considering the results reported, the conclusion is that the RAS model is clearly inappropriate for this study, since it tends to over-estimate the stability properties of our flow. It is understandable considering the fact that a time average motion will cut out all small scales turbulence, which can be considered as a fluctuating phenomena. We aren't trying to prove that RAS is generally a bad approach: indeed it has some great advantages, like the short execution time of simulations, but, if one is interested in a detailed study of a given turbulent phenomenon, this is not the approach to follow.

2.4 Large Eddy Simulations

Like RAS, Large Eddy Simulations try to save time in solving turbulent flows by limiting the number of elements in the mesh and modeling turbulence in another way. One drawback with respect to RAS is their high computational cost, balanced out by their better accuracy, as we will verify later in our work.

2.4.1 The filtering process

In case of LES, all information pertaining to the smallest scales is cut out from the solution by *low-pass filtering* all the equations[13]. In general, a *low-pass filter* is an integral operator which acts on fields and deletes all of its components with frequency higher than a certain *cutoff frequency* (imagine to write it as a Fourier series...). It has the following form:

$$\widetilde{f}(\vec{x},t) = \int_V G(\vec{x},\vec{x}',\Delta) f(\vec{x}',t)\, d\vec{x}' \tag{2.8}$$

$\vec{x}$ is the point where $\tilde{f}$ is evaluated, while $\vec{x}'$ goes through all the domain in order to gather values of f in points relevant to the filtering process[14]. This selection is carried out thanks to the *filter function* G, which must satisfy the following property:

$$\int_V G(\vec{x},\vec{x}',\Delta)\, d\vec{x}' = 1 \tag{2.9}$$

This suggests us that suitable filter functions are those which

- are limited in maximum and minimum value,
- tend to 0 when moving towards boundaries or have compact support.

The level of filtering is then set by the parameter Δ. In terms of integration, Δ decides how far from $\vec{x}$ the relevant points will be: the higher its value, the bigger the region of interest and the lower the cutoff frequency, which is $1/\Delta$. The filter effect on a generic function is reported in Figure 2.8:

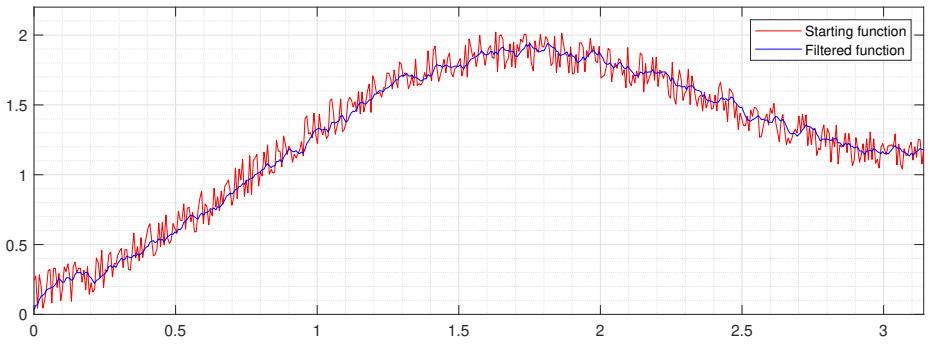


Figure 2.8. *Effect of filtering on a generic one-dimensional function.*

[13] In our case, we won't apply the filter to (1.14), since it would require a deep modification of the source code of OpenFOAM. Nonetheless, as we will see, our solutions are satisfactorily accurate.

[14] In our case, the filtering operation is only in space.

Another important feature which may characterize a filter function is *homogeneity*: G is *homogeneous* if

$$G = G(\vec{x}', \Delta) \tag{2.10}$$

This property practically translates in the filtering operation not depending on the point $\vec{x}$ where filtering occur. It is crucial if one wants to commute filter operator with spatial differential operators. It is thus essential for our purpose.
By applying the filter to the system, we will get a set of equations where the filtered quantities are the new variables. However, The system is not close yet due to a new term, which arises from the filtered (1.3):

$$\overline{\overline{\tau}}_{SGS} \equiv \widetilde{\vec{u} \otimes \vec{u}} - \widetilde{\vec{u}} \otimes \widetilde{\vec{u}} \tag{2.11}$$

$\overline{\overline{\tau}}_{SGS}$ is called *sub-grid scale stress tensor* $(\mathrm{m}^2\,\mathrm{s}^{-2})$[15]. As its name suggests, it is the term which holds information pertaining to the filtered scales (together with α_{eff} in (1.21)[16]). We believe it's interesting to explain in which sense the memory of lower scales is kept inside $\overline{\overline{\tau}}_{SGS}$. Firstly, we need to define *large scales* the ones with dimension bigger than Δ and *small scales* the one with dimension lower than Δ. The information inside product $\vec{u} \otimes \vec{u}$ can be divided into four contributions:

1. how large scales influence large scales,
2. how large scales influence small scales,
3. how small scales influence large scales,
4. how small scales influence small scales.

Via filtering process one gets rid of all information contained in small scales, thus leaving only points 1 and 3. Finally, by subtracting $\widetilde{\vec{u}} \otimes \widetilde{\vec{u}}$, one is left with only point 3, which is the most important contribution[17]: since all scales above Δ are directly solved, we will preserve the influence of all filtered scales inside our solution.

2.4.2 Smagorinsky model

Now, in order to close the system, it is fundamental to introduce a way to model (2.11). One of the most famous approaches comes from Smagorinsky (1963), whom firstly considered the decomposition of $\overline{\overline{\tau}}_{SGS}$ into its anisotropic and isotropic part. For the sake of clarity, we will adopt Einstein notation in the following dissertation.

$$\tau_{ij}^{SGS} = \tau_{ij}^{an} + \frac{1}{3}\tau_{kk}^{SGS}\delta_{ij} \tag{2.12}$$

[15] Even though it doesn't have the dimensions of a stress, it is usually called in this way in literature.

[16] What has been explained in footnote 10 holds also here. Clearly the method to assign a value to α_{eff} is different between RAS and LES, but it is beyond our scope to address this specific topic. Generally it is more interesting to study the terms reported in (2.5) and (2.11), since they bring some physical meaning with them.

[17] Information on how large scales influence large scales is already contained inside the advection term.

where δ_{ij} is the Kronecker delta. He then added the isotropic part to the pressure variable. Regarding the anisotropic part, he proposed the following:

$$\tau_{ij}^{an} = -2\nu_T \widetilde{S}_{ij} \tag{2.13}$$

$$\widetilde{S}_{ij} \equiv \frac{1}{2}\left[\left(\frac{\partial \widetilde{u}_i}{\partial x_j} + \frac{\partial \widetilde{u}_j}{\partial x_i}\right) - \frac{2}{3}\frac{\partial \widetilde{u}_k}{\partial x_k}\delta_{ij}\right] \tag{2.14}$$

where $\widetilde{S}_{ij}$ is the *strain rate tensor* (s^{-1}). Thanks to this approach, the anisotropic part can be treated together with the viscous stress tensor, modifying the viscosity of the fluid[18]. The problem is now shifted towards finding a model for ν_T.
Via dimensional analysis, one can get to:

$$\nu_T \propto lv \tag{2.15}$$

where l and v are, respectively, characteristic length and velocity of sub-grid scales. Since resolved scales are mostly influenced by the largest unresolved ones, one can assume:

$$l = C_S \Delta \tag{2.16}$$

C_S is an empirical parameter called *Smagorinsky constant*. It can be found by means of benchmark simulations. Concerning v, Prandtl's mixing length theory (1925) suggests the following:

$$v = l|\widetilde{S}| = C_S \Delta |\widetilde{S}| \tag{2.17}$$

$$|\widetilde{S}| \equiv \sqrt{2\widetilde{S}_{ij}\widetilde{S}_{ij}} \tag{2.18}$$

Kinetic turbulent viscosity can thus be expressed as:

$$\nu_T = C_S^2 \Delta^2 |\widetilde{S}| \tag{2.19}$$

This model is relatively simple to implement, but it has some deficits. One for sure consists in overestimating turbulence close to walls: indeed in that region velocity has strong gradients, so $|\widetilde{S}|$ will increase and, as a consequence, ν_T will increase too. This is in contrast with the physics of our problem, where turbulence dies out close to walls due to viscous dissipation. Van Driest, as part of the mixing length RAS model (1956), introduced an approach which has been applied also in the present case:

$$l = C_S \Delta \left(1 - e^{\frac{y^+}{A^+}}\right) \tag{2.20}$$

where y^+ is the normalized distance from the wall and A^+ is an empirical parameter (usually set to 25). Nevertheless, it has been proven that results were not always as expected. Moreover the Smagorinsky model tends to become inaccurate in cases where laminar/turbulent transitions and viceversa occur, as in our simulations, where we want to predict laminarization of the flow. Another issue is linked to the determination of C_S: it usually ranges from 0.065 to 0.165, depending on the case, so we should spend quite a long time figuring out the right value.

[18]The term appears in the equation multiplied by $\widetilde{\rho}$, so it has the same dimension of the viscous stress tensor.

2.4.3 WALE model

The *Wall Adaptive Local Eddy-viscosity* (WALE) model is a more recent approach to our problem, which has been proven to perform well in many cases. It implies the following formulation for ν_T:

$$\nu_T = C_k \Delta \sqrt{\kappa_{SGS}} \tag{2.21}$$

$$\kappa_{SGS} \equiv \frac{1}{2}\tau_{kk}^{SGS} = \frac{1}{2}(\widetilde{u_{kk}u_{kk}} - \widetilde{u}_{kk}\widetilde{u}_{kk}) \tag{2.22}$$

κ_{SGS} is the *sub-grid scale turbulent kinetic energy* ($\mathrm{m}^2\,\mathrm{s}^{-2}$), which has been formulated as follows[19]

$$\kappa_{SGS} = \left(\frac{C_w^2\Delta}{C_k}\right)^2 \frac{(\widetilde{S}_{ij}^d \widetilde{S}_{ij}^d)^3}{\left((\widetilde{S}_{ij}\widetilde{S}_{ij})^{\frac{5}{2}} + (\widetilde{S}_{ij}^d \widetilde{S}_{ij}^d)^{\frac{5}{4}}\right)^2} \tag{2.23}$$

where $\widetilde{S}_{ij}^d$ is the traceless symmetric part of $\widetilde{S}_{ij}$. Combining (2.21) and (2.23), the final formulation of ν_T will be:

$$\nu_T = (C_w\Delta)^2 \frac{(\widetilde{S}_{ij}^d \widetilde{S}_{ij}^d)^{\frac{3}{2}}}{(\widetilde{S}_{ij}\widetilde{S}_{ij})^{\frac{5}{2}} + (\widetilde{S}_{ij}^d \widetilde{S}_{ij}^d)^{\frac{5}{4}}} \tag{2.24}$$

As we can see, the WALE model is able to maintain the simplicity of Smagorinsky, since it doesn't require the addition of new equations to the problem[20]. Moreover, it has been shown by different authors that it can solve more precisely both turbulence damping at walls and laminarization of the flow with respect to Smagorinsky. For such reasons, we will adopt the WALE model in our simulations.

2.4.4 OpenFOAM case setup

Regarding case setup, we can safely refer to Section 2.2, since a significant comparison between LES and no-model can be achieved only by adopting the same settings. We will limit ourselves to show only those specific to LES.
One fundamental feature of such kind of simulation is its three-dimensional nature. Even though large eddies are strongly influenced by the geometry (see Figure 2.2) small scale turbulence tends to behave independently from that. For this reason a 3D setup is necessary, so that filtered scales can correctly influence resolved scales.

0/nut

When setting up LES simulation in OpenFOAM, it's necessary to declare boundary and initial conditions for ν_T, as if it was an actual field.

```
dimensions      [0 2 -1 0 0 0 0];

internalField   uniform 0;
```

[19] We are not interested in showing how to obtain such formulation, since it would be a very long process.

[20] Other LES models require such addition, similarly to the $\kappa - \omega$ model introduced in 2.3.1.

```
boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 0;
    }

    outlet
    {
        type            zeroGradient;
    }

    upperWall
    {
        type            fixedValue;
        value           uniform 0;
    }

    lowerWall
    {
        type            fixedValue;
        value           uniform 0;
    }

    front
    {
        type            symmetryPlane;
    }

    back
    {
        type            symmetryPlane;
    }
}
```

Firstly, it's important to introduce the `symmetryPlane` type boundary condition. Essentially, it simulates the continuation of the geometry by acting like a symmetry plane between the real geometry and the simulated one. In this way our fluid will act as if upper and lower walls elongate over the boundaries. A proper definition of `front` and `back` in `blockMeshDict` (under `boundary` keyword) is required in order to use this boundary condition:

```
front
{
    type    symmetryPlane;
    faces
```

```
    (
            (0 7 6 1)
            (1 6 5 4)
            (2 1 4 3)
    );
}

back
{
    type    symmetryPlane;
    faces
    (
            (8 9 14 15)
            (9 12 13 14)
            (10 11 12 9)
    );
}
```

The reason behind such choice is the three-dimensional nature of LES simulations. Regarding the initial condition, it is coherent with the fact that our fluid is motionless at time 0, so no turbulence can be present. The same holds for upper and lower wall boundary conditions: the fluid remains still near walls due to the no-slip, so no turbulence develops there. Regarding the inlet, since the fluid enters with a uniform velocity, it is reasonable to assume that what just stated about walls holds also here. Finally, at the outlet we figured out that `zeroGradient` is the most suited condition, since there is no physical reason for turbulent phenomena to change behavior while exiting the geometry. Moreover, it is coherent with the velocity field.

0/alphat

Together with ν_T, OpenFOAM requires the user to specify conditions also for the turbulent contribution to α_{eff}, which goes under the name α_T. Since its value depends on the presence of turbulence, it is advisable to set all its conditions equal to those of ν_T.

turbulenceProperties

As already mentioned, here is where we declare the nature of our simulation.

```
simulationType  LES;

LES
{
    LESModel            WALE;

    turbulence          on;

    printCoeffs         on;
```

```
    delta               cubeRootVol;

    cubeRootVolCoeffs
    {
        deltaCoeff      1;
    }
}
```

The code is straightforward. Indeed, the only entry which needs clarification is `cubeRootVol`: it implies that our Δ will be the cubic root of the cell volume. This is a very convenient option, since it allows the user to reduce filtering level, thus increasing accuracy of the solution, by directly refining the mesh in the regions of most interest. Another important keyword is `deltaCoeff`. As its name suggests, it's a coefficient that multiplies Δ, so, if one wants to refine at different scales with respect to cell dimension, it is sufficient to set it to a value lower o higher than 1, respectively if more or less accuracy are desired.

2.4.5 Results

Before proceeding with our results, the mesh which will be mostly used in the following will be introduced. Its dimensions are the same of figure 2.4, plus 1 (cm) in thickness, so that the inlet patch will be a square. Regarding number of elements and grading, we refer to the following code:

```
blocks
(
    hex (0 1 6 7 8 9 14 15) (120 36 36) simpleGrading (0.2 1 1)
    hex (2 3 4 1 10 11 12 9) (290 36 36) simpleGrading (5 1 1
    hex (1 4 5 6 9 12 13 14) (290 36 36) simpleGrading (5 1 1)
);
```

which brings to a total of 907200 elements. As our first approach to the problem, we preferred to limit the number of elements, so that simulations won't last too much, allowing us to check if all settings were appropriate.
Unfortunately, we cannot resort to the stability analysis we previously introduced in this chapter, since it was only developed for 2D simulations.

Laminarization time vs Ha

Firstly, we analyzed the laminarization time with respect to Ha value, with the aim of finding at which value it becomes laminar in the limit of 8 seconds.

Ha	Laminarization time (s)
24	Not in time limit
25	Not in time limit
26	Not in time limit
27	3.6
28	3.6
30	2.4
35	0.6
40	0.4
60	0.04
80	0.08
100	0.09

Table 2.4. *Comparison between laminarization time for various Ha values.*

We can notice that for $Ha = 24$ stabilization is not yet reached, which is the same result we obtained for the finest mesh (table 2.2). This fact is very positive, since it implies that unresolved scales are correctly influencing solved ones.
For the sake of a clear comparison, we will now plot the data, showing also a fitting curve of our results.

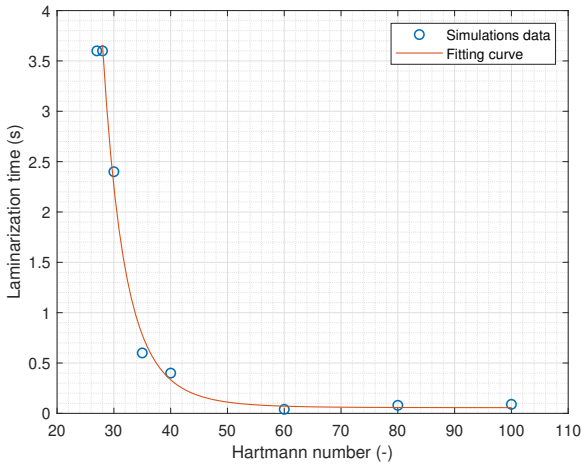


Figure 2.9. *Hartmann number vs laminarization time.*

The law which best fits our number is the following:

$$t_l(Ha) = 9.38 * 10^{10} Ha^{-7.197} + 0.05656 \tag{2.25}$$

where t_l stands for laminarization time. In order to get a clean interpolation, we excluded $Ha = 27$ from the data. Indeed, for Ha between 26 and 28, more simulation could reveal a different trend with respect to the interpolating curve, which would still remain correct in case of $Ha \geq 28$. The problem is that such analysis would be very time-consuming since it would require a large number of simulations[21].
Before proceeding, we must mention the fact that a different mesh was necessary in

[21] It could be addressed in a future development of our work.

order to simulate $Ha = 60, 70, 80$. Indeed in such cases the Hartmann layer is really thin. This results in a very steep velocity gradient close to walls, which requires an adequate mesh refinement[22]. Placing five cells inside the layer is enough to ensure that the fluid behavior is correctly reproduced.

LES vs no-model finest mesh

Aware of the aforementioned results, we are now interested in a direct comparison between no-model finest mesh and LES. We will consider $Ha = 27$, since it's the lowest value which assures stabilization. A comparison between no-model profiles and LES profiles taken 6 cm away from the step will be reported in Figure 2.10.

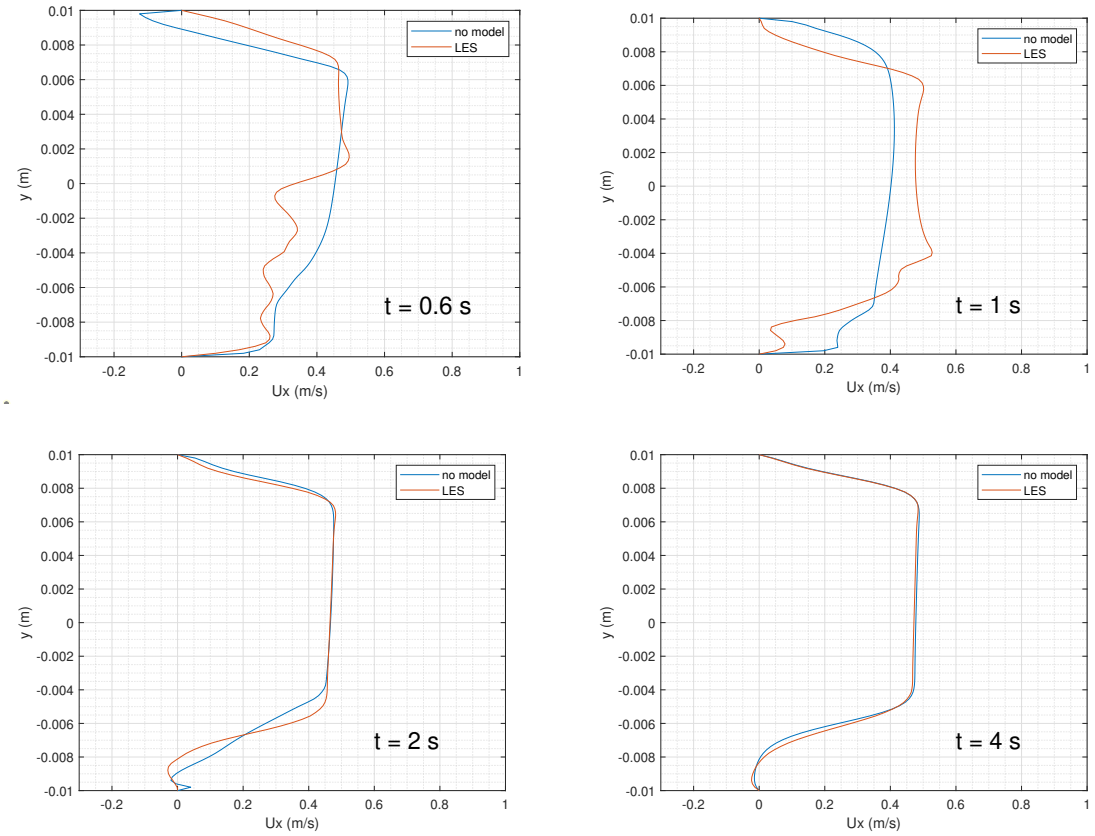


Figure 2.10. *Comparison between u_x profile taken* 6 cm *away from the step without model and with LES model.*

From the profiles, it emerges that LES predicts turbulence slightly differently from the no-model case, while the final steady state profiles show good agreement. The difference in the turbulent behavior can be motivated sureby by the employment of LES model, but also considering the 3D nature for such simulation, even though its impact on the velocity field should be investigated.
Three different aspects are then evaluated: laminarization time, number of elements per square centimeter of mesh surface[23] and simulation execution time. They are

[22] We noticed an unphysical increase of $\vec{u}$ close to walls before adopting a sufficiently refined mesh.
[23] This is the clearest way to compare mesh refinement between 2D and 3D simulations. Moreover, our case can be approximated as two-dimensional, so our method is even more adequate.

gathered in Table 2.5.

Simulation type	Laminarization time (s)	elements/cm^2	Execution time (s)
no model	3.8	18670	281524
LES	3.6	360	84225

Table 2.5. *Comparison between finest mesh and LES on laminarization time, number of elements per square centimeter of mesh surface and simulation execution time.*

The most relevant result is surely the fact that both predicted similar laminarization times, but with a huge difference in elements/cm^2, confirming the capability of LES to correctly reproduce filtered scales influence on resolved scales. Moreover, our LES simulation was roughly 3.3 times faster than the other. The reason behind such gap is to be searched inside the Courant number (1.31): in order to satisfy the restrain $Co \leq 0.9$, time step must adapt itself to the mesh size, which is noticeably smaller for the finest mesh.

Three-dimensional effects

The impact of u_z on the development of the flow will be now addressed. The profile of the three components of $\vec{u}$ will be compared along two sections: one 6 cm and one 27 cm far from it. We will employ $Ha = 27$ also for this case.

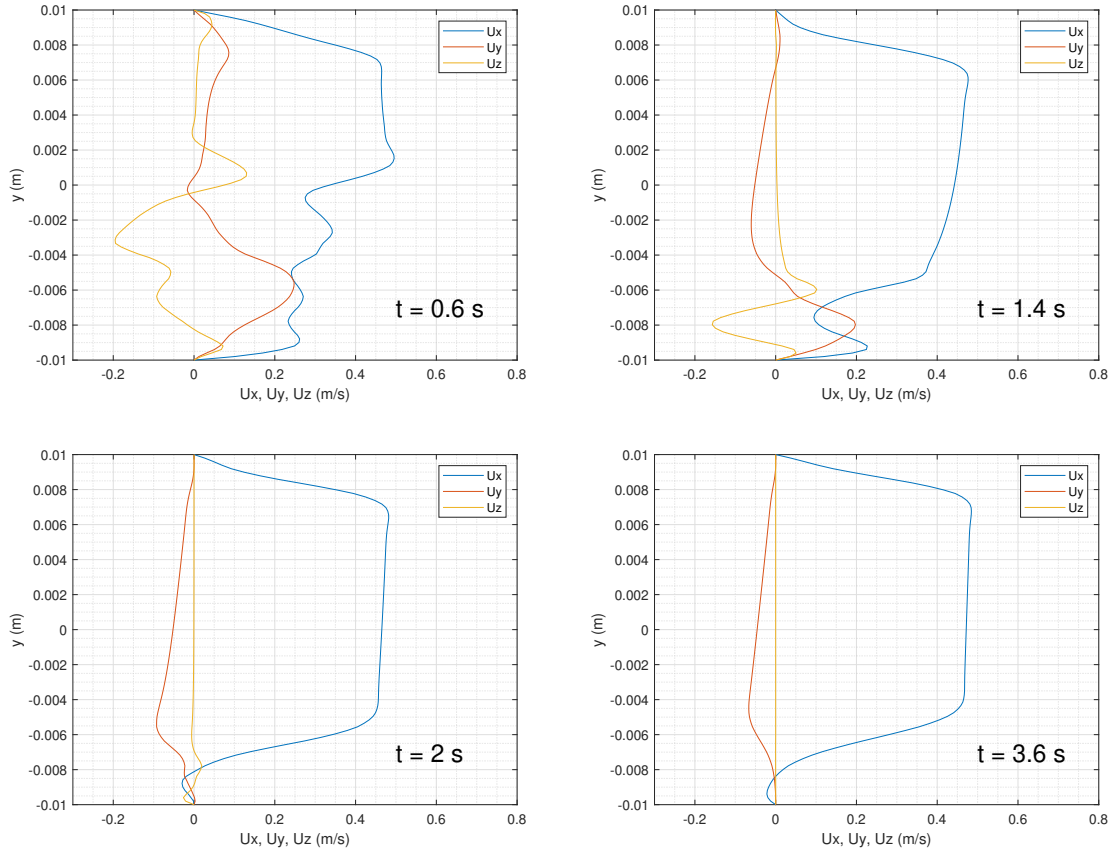


Figure 2.11. *Profiles of u_x, u_y and u_z taken* 6 cm *away from the step.*

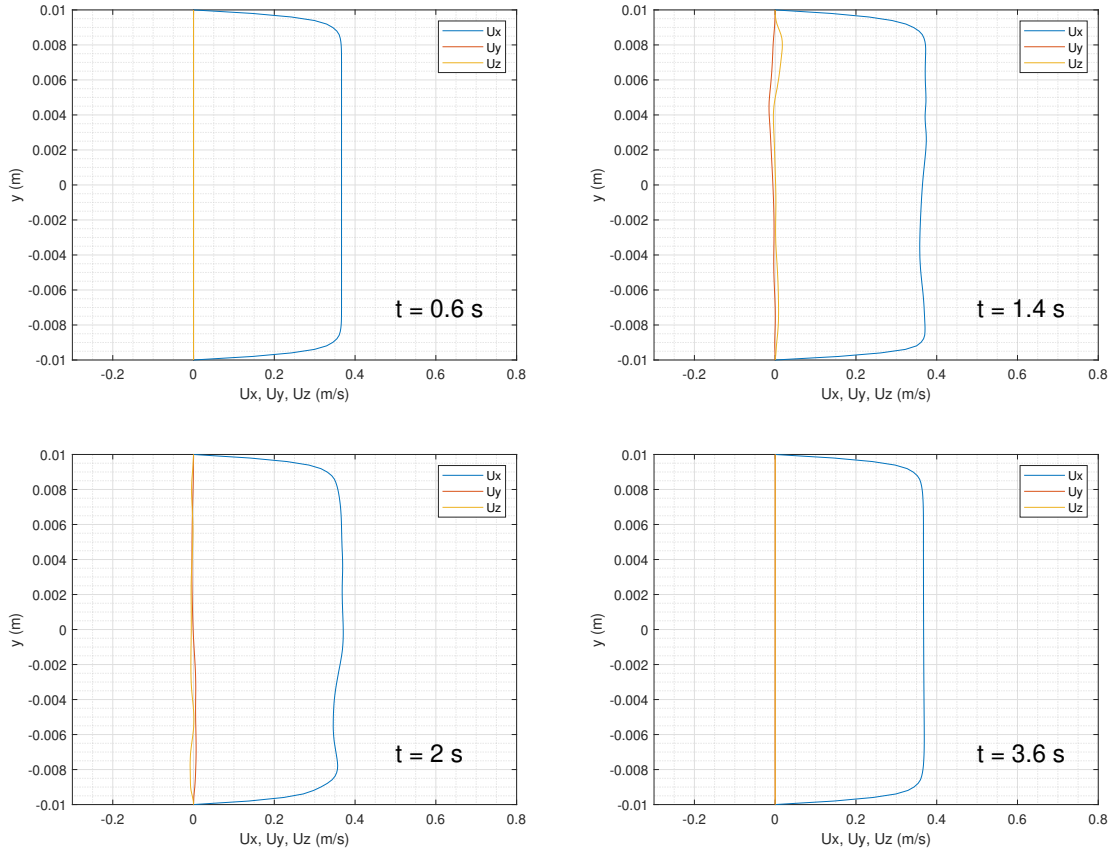


Figure 2.12. *Profiles of u_x, u_y and u_z taken* 27 cm *away from the step.*

The most interesting results are surely the ones pertaining to the section closer to the step, where the recirculating region develops. As we can see, in that region u_z assumes values which are comparable to those of u_x and u_y. Advancing in time or moving away from the step lowers its importance. Indeed, by advancing in time the effect of $\vec{B}$ becomes more relevant, thus bringing the flow to its two-dimensional nature[24], while keeping the distance from the recirculating region results in a less chaotic behavior from the start.
In conclusion, a three-dimensional simulation is sensibly more accurate than a two-dimensional one if the interest lies in transient analysis. It is recommended to resort to 2D only if there is certainty that $\vec{B}$ will laminarize the flow and if the interest is only in the steady state profiles.

Reattachment length

The *reattachment length* will be analysed next. We will consider the same Ha values reported in Table 2.4 and report all data in Table 2.6

[24]Indeed, the action of $\vec{B}$ is to shape u_x as the Hartmann analytical profile (1.22).

Ha	Reattachment length (cm)
24	5.5 ÷ 7
25	5.5 ÷ 7
26	6 ÷ 7
27	6.6
28	6.3
30	5.7
35	4.6
40	3.7
60	2.2
80	1.6
100	1.2

Table 2.6. *Comparison between laminarization time for various Ha values.*

As we could expect, for Ha value corresponding to absence of stabilization it's impossible to obtain a fixed value for r. In those cases, we will limit ourselves to report the range of values in which r oscillates.
In the wake of what we did for the laminarization time, we will now gather all data in a plot and fit them.

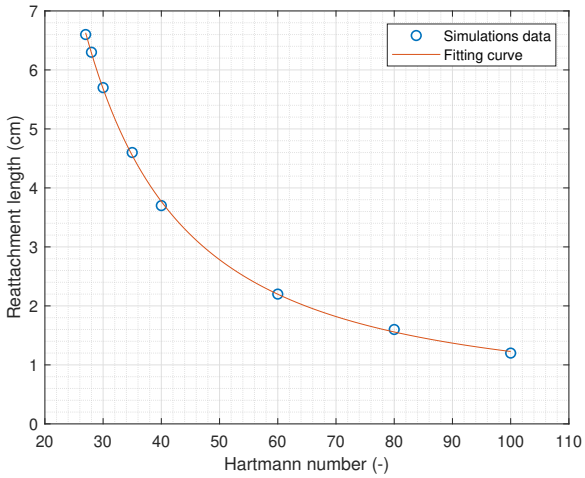


Figure 2.13. *Hartmann number vs reattachment length.*

The law followed by $r(Ha)$ is similar to (2.25):

$$r(Ha) = 1124Ha^{-1.579} + 0.4442 \tag{2.26}$$

As we can see, both (2.25) and (2.26) are power laws, even though coefficients are quite different. By looking at figures 2.9 and 2.13, it's noticeable how the latter results more accurate than the former. Indeed the study of r is simpler to carry out, since the only requirement is the achievement of a steady state, while the accuracy of t_l could be influenced by the amount of time between two consecutive data sampling, for which the storage capability is a limiting factor.

Mesh sensitivity analysis on LES

In conclusion of the second chapter, we will undergo a brief mesh sensitivity analysis on LES, considering just one refinement. It's important to notice that in this case the refinement process translates into reducing Δ. Since we already performed various simulations on the mesh presented at the beginning of 2.4.5, we will select a value for Ha and perform a new simulation on a refined version of that same mesh. We will consider once again $Ha = 27$, since it's the most significant example in terms of laminarization time. Figure 2.14 shows u_x profiles taken 6 cm away from the step, while Table 2.7 gathers some interesting data: number of cells in the mesh, execution time of the simulation, laminarization time and reattachment length.

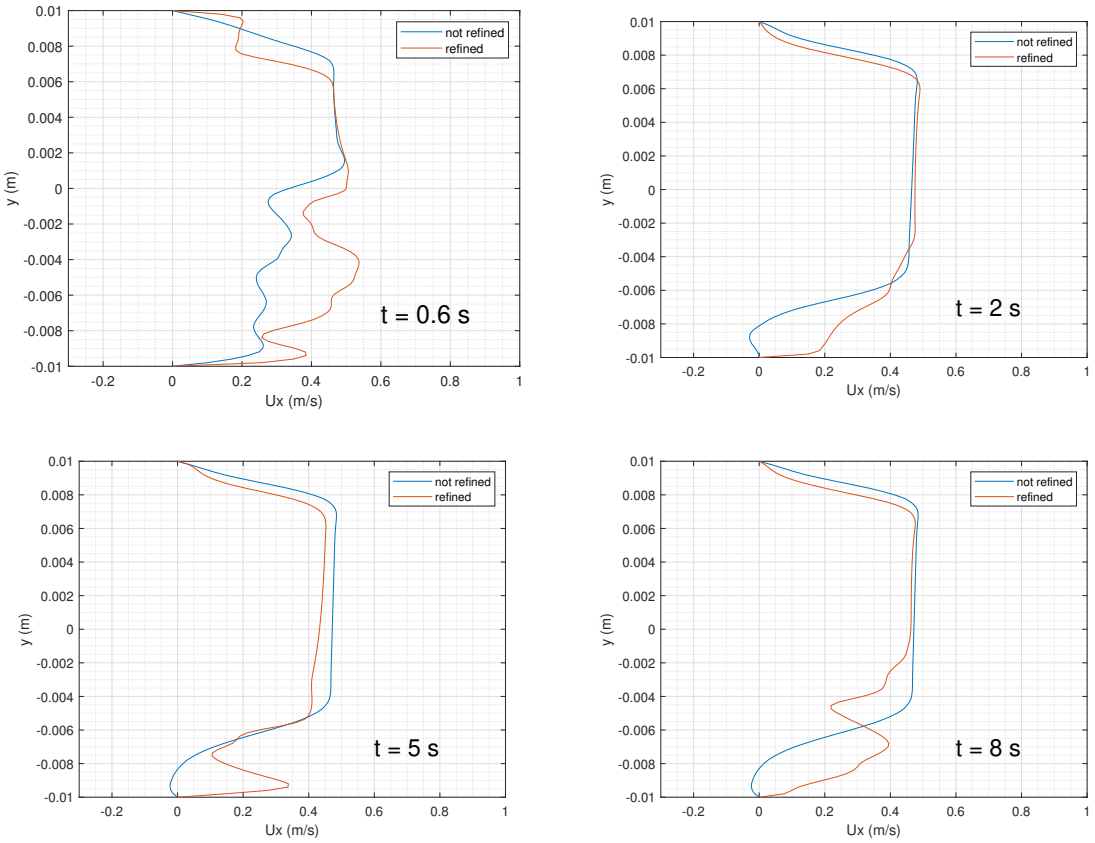


Figure 2.14. *Comparison between u_x profiles taken* 6 cm *away from the step without and with mesh refinement.*

mesh	total number of cells	execution time (s)	t_l (s)	r (cm)
standard	907200	84225	3.6	6.6
finer	1782000	506536	Not in time limit	$6 \div 7$

Table 2.7. *Comparison between LES on two different meshes.*

As we can see, approximately doubling the number of elements results in a sensibly more chaotic motion and in the absence of laminarization in the time limit of 8 seconds. Considering Figure 2.9, we can assume that the effect of mesh refinement is to shift the curve to the right by a certain amount, till the point in which further refining

brings to the same result[25].
It has to be said that such accurate analysis turned out to be very expensive in terms of execution time. For this reason, one should always consider if it's worth investing this large amount of time in one single simulation.

2.5 Conclusions

In the present chapter, we were interested in testing out the capability of comprMhdFoam to employ turbulence models in MHD problems. The first step was the mesh sensitivity analysis, which was necessary in order to have a reference point later on. Secondly, RAS and LES models were introduced and tested. The latter brought to interesting results, which encouraged us to investigate some features of the physical system.
What has been learned is that the best way to approach such problem is to consider a three-dimensional geometry and LES turbulence model. In this way, the characteristic of the system in terms of dynamical behavior can be addressed. Once those aspects are understood, one can decide to switch to a faster approach (for example RAS), especially if his interest lies in the steady state after the transient.
Another aspect which emerged is the sensitivity of LES to mesh refinement: indeed, the more the mesh is refined, the more the solution tends to the one of a DNS, since the filter length Δ cuts less scales out of the turbulence field. So, one should always decide which grade of precision is enough for his purpose, maybe by meas of a comparison with experimental results. Given the amount of time which just one simulation can take, one could also be interested in a way to speed up calculations. Such objective can be achieved by employing a dimensionality reduction algorithm. For this reason, *Dynamic Mode Decomposition* (DMD) will be introduced and applied to some of the simulations we carried out during this chapter, aiming at determining if it could be a viable approach to save computational time.

[25]The more one refines, the closer he gets to a Direct Numerical Simulation.

Chapter 3

Dynamic Mode Decomposition approach

In the last section of this thesis work, we will apply the *Dynamic Mode Decomposition* approach to some of our simulations in order to obtain a reduced-order model of our system. Firstly, we will theoretically introduce the algorithm, then we will proceed to explain how we implemented it in Matlab, where in a single script we gather data from the snapshots of our simulations and employ them as starting point for all the necessary computations. Finally, we will analyze the results and declare whether this approach could be viable or not to speed up future simulations.

3.1 Introduction

Given the fact that several simulations are usually necessary to fully understand any addressed phenomenon and given the large amount of time which just one of them can take, it is reasonable to think of a way to save such time. Indeed, as addressed in Subsection 2.4.5, if one wants to obtain highly precise results, simulations will tend to last for days at best. In these kind of situations, a technique called *Dynamic Mode Decomposition* (DMD) can find an interesting application.
As explained by [Di Ronco et al., 2020], DMD algorithm aims at reconstructing the dynamics of a given dynamical system through a linear combination of empirical state vectors called *modes*. It is a *spatial dimensionality reduction* technique in the sense that, depending on the problem at hand, the number of basis vectors required to capture the dominant dynamics of the system can be several orders of magnitudes smaller than the original problem size. One very important feature which makes this approach interesting is the fact that it is equation-free: indeed what is needed by the algorithm to work are only snapshots containing fields data (velocity for example) taken every fixed amount of time. These snapshots can be either taken from a simulation or from an actual experiment through measurements. This directly implies that the dynamics pertaining to our system can also be non-linear.
As mentioned by [Kutz et al., 2016], DMD is mostly employed in three different situations:

- **Diagnostic**: DMD allows for the data-driven discovery of fundamental, low-rank structures in complex systems, such as hydrodynamics ones.

- **State estimation and future-state prediction**: unlike the diagnostic objective, here the goal is to anticipate the state of the system in a regime where no measurements were made. This is a much more difficult task, especially as DMD is limited to constructing the best-fit (least-square) linear dynamical system of the non-linear dynamical system generating the data.
- **Control**: enabling viable and robust control strategies directly from data sampling is the ultimate, and most challenging, goal of the DMD algorithm. Given the fact that we are approximating a non-linear behavior with a linear one, it is reasonable to assume that the prediction of the future of our system would be accurate only for a limited amount of time. The hope is that such period is long enough to allow for good decision-making about the state of the system.

Concerning our purpose, we will start from snapshots of a LES simulation and aim at reconstructing them by employing DMD, which is the first and most simple task when approaching such problem. A key factor will be the time step between subsequent snapshots: its optimal value must be found through trial and error if the characteristic times of the dynamic are not clearly known. Indeed, if it's too long, short time dynamical events can't be caught, while, if it's too short, the long time ones get lost. We will thus study the difference emerging by considering different time steps.

3.2 The DMD algorithm

The algorithm starts with the gathering of all the snapshot of our dynamical system, which has dimension n, as a matrix composed of m column vectors of length n, each one corresponding to the system state at a given time.

$$\mathbf{X}_1^m = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m\} \tag{3.1}$$

where each $\mathbf{x}_k \in \mathbb{R}^n$, $k = 1, 2, ..., m$, is taken at a fixed Δt with respect to the previous one. For the success of the algorithm it is important that these Δt are identical between each other or at least very similar[1]. In our case, each vector $\mathbf{x}$ contains the values assumed by one of our fields across the entire mesh at a given time, which brings to one number for each cell in case of a scalar fields. In case of vector fields, in order to maintain the structure of $\mathbf{X}_1^m$, it is reasonable to organize $\mathbf{x}$ as follows: the first third will be the x component, the second third the y component and the last one the z. Since we are using the same mesh presented at the beginning of 2.4.5 and are interested in reconstructing the velocity field, in our case $n = 907200 * 3 = 2721600$. Since we are dealing with a discrete problem, we can now assume the existence of a linear map $\mathbf{A}$[2] which can advance $\mathbf{x}$ in time:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k \tag{3.5}$$

[1] As we will see, in our case this condition isn't fully accomplished since we use an adaptive time step, but the difference between them is negligible.

[2] A generic dynamical system can be represented by the following:

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, t; \mu) \tag{3.2}$$

The DMD algorithm produces a low-rank eigendecomposition of the matrix A that optimally fits the data x_k for $k = 1, 2, ..., m$ in a least-square sense so that

$$||\mathbf{x}_{k+1} - \mathbf{A}\mathbf{x}_k||_2 \tag{3.6}$$

is minimized across all points for $k = 1, 2, ..., m-1$. The optimality of the approximation holds only over the sampling time window in which snapshots are gathered and $\mathbf{A}$ is constructed. Another way to see (3.5) is the following:

$$\mathbf{X}_2^m = \mathbf{A}\mathbf{X}_1^{m-1} \tag{3.7}$$

from which $\mathbf{A}$ can be derived as follows:

$$\mathbf{A} = \mathbf{X}_2^m \left(\mathbf{X}_1^{m-1}\right)^\dagger \tag{3.8}$$

where the † superscript denotes the *Moore-Penrose pseudoinverse*[3].
The problem is that our n is really large[4], so a direct analysis of $\mathbf{A}$ could take a very long amount of time. What DMD does is to circumvent the eigendecomposition of A by considering a rank-reduced representation in terms of a POD[5]-projected matrix $\hat{\mathbf{A}}$. Now, the algorithm itself will be introduced step-by-step.

1. Firstly, a *Singular Value Decomposition*[6] (SVD) of $\mathbf{X}_1^{m-1}$ is performed:

$$\mathbf{X}_1^{m-1} \approx \mathbf{U}\mathbf{S}\mathbf{V}^* \tag{3.9}$$

   where $*$ superscript denotes the conjugate transpose, $\mathbf{U} \in \mathbb{C}^{n\times(m-1)}$, $\mathbf{S} \in \mathbb{C}^{(m-1)\times(m-1)}$ and $\mathbf{V} \in \mathbb{C}^{(m-1)\times(m-1)}$. The columns of $\mathbf{U}$ and the columns of $\mathbf{V}$ are called the *left-singular vectors* and *right-singular vectors* of $\mathbf{X}_1^{m-1}$,

where $\mathbf{x}$ is the state vector of our system at time t, while μ is the set of parameters involved. Since in general a dynamical system is described by a coupled system of ordinary, often non-linear differential equations, it is usually not possible to construct a solution to the governing non-linear evolution. What DMD wants to accomplish is the construction of the proxy, approximate locally linear dynamical system

$$\frac{d\mathbf{x}}{dt} = A\mathbf{x} \tag{3.3}$$

starting from the snapshots of the non-linear problem, which dynamics can remain unknown. Given the continuous problem 3.3, it is always possible to describe an analogous discrete-time system sampled every Δt in time. The bond between (3.5) and (3.3) is the following:

$$\mathbf{A} = exp(A\Delta t) \tag{3.4}$$

We will thus aim at reconstructing $\mathbf{A}$, from which the continuous, approximate linear dynamics can be retrieved.

[3]The *Moore-Penrose pseudoinverse* is a generalization of the inverse matrix. It is equivalent to finding the best-fit solution (in the least-squares sense) of a system of linear equations lacking a unique solution.

[4]Left apart our case, n is usually very large.

[5]The *Proper Orthogonal Decomposition* (POD) method essentially provides an orthogonal basis for representing a given set of data in a least-squares optimal sense, i. e., it offers ways to find optimal lower-dimensional approximations for the given data set.

[6]The *Singular Value Decomposition* is a factorization of a real or complex matrix that generalizes the eigendecomposition of a square normal matrix to any $n \times m$ matrix via an extension of the polar decomposition.

respectively. Moreover, the columns of $\mathbf{U}$ are POD modes and are orthonormal, so $\mathbf{U}^*\mathbf{U} = 1$; similarly, $\mathbf{V}^*\mathbf{V} = 1$.
Among the three matrices, the most important entry is $\mathbf{S}$: it is diagonal and its elements are called the *singular values* of $\mathbf{X}_1^{m-1}$. Basically, the higher a singular value is, the more relevant its associated mode will be, so, by looking at $\mathbf{S}$, one can determine which and how many of them will be necessary to properly reconstruct our field. Since the SVD is not unique, it is always possible to choose the decomposition so that the singular values S_{ii} are in descending order. In this way the truncation of undesired modes is achieved just by reducing the dimensions of our three matrices accordingly to the number of desired modes $r < m - 1$: $\mathbf{U} \in \mathbb{C}^{n\times r}$, $\mathbf{S} \in \mathbb{C}^{r\times r}$ and $\mathbf{V} \in \mathbb{C}^{m\times r}$.

2. Now we can reconstruct $\mathbf{A}$ by considering the pseudoinverse of $\mathbf{X}_1^{m-1}$ obtained via SVD:
$$\mathbf{A} = \mathbf{X}_2^m \mathbf{V}\mathbf{S}^{-1}\mathbf{U}^* \tag{3.10}$$
In practice the storage of $\mathbf{A}$ can be problematic (in our case it would be a 2721600×2721600 matrix), so it is much more convenient to compute $\hat{\mathbf{A}}$, which is its projection onto POD modes:
$$\hat{\mathbf{A}} = \mathbf{U}^*\mathbf{A}\mathbf{U} = \mathbf{U}^*\mathbf{X}_2^m\mathbf{V}\mathbf{S}^{-1} \tag{3.11}$$
The matrix $\hat{\mathbf{A}}$ defines a low-dimensional linear model of the dynamical system on POD coordinates:
$$\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{A}}\hat{\mathbf{x}}_k \tag{3.12}$$
while the full-dimensional state can be recovered by mean of $\mathbf{U}$:
$$\mathbf{x}_k = \mathbf{U}\hat{\mathbf{x}}_k \tag{3.13}$$

3. The reconstruction of $\hat{\mathbf{A}}$ is followed by its eigendecomposition:
$$\hat{\mathbf{A}}\mathbf{W} = \mathbf{W}\mathbf{L} \tag{3.14}$$
where the columns of $\mathbf{W}$ are eigenvectors and $\mathbf{L}$ is a diagonal matrix containing the corresponding eigenvalues λ_i. We can now compute the eigenvalues of the continuous problem:
$$\omega_i = \frac{\ln(\lambda_i)}{\Delta t} \tag{3.15}$$

4. From $\mathbf{W}$ and $\mathbf{L}$ we can reconstruct the eigenvectors of $\mathbf{A}$, which will be the columns of $\mathbf{P}$:
$$\mathbf{P} = \mathbf{X}_2^m\mathbf{V}\mathbf{S}^{-1}\mathbf{W} \tag{3.16}$$
It has been proven that the eigenvectors defined by (3.16) are the exact eigenvectors of $\mathbf{A}$.

5. Finally, we can write the reconstruction of our field for any given time in the future:
$$\mathbf{x}(t) \approx \sum_{i=1}^{r} \mathbf{p}_i \exp(\omega_i t) b_i = \mathbf{P}\exp(\mathbf{O}t)\mathbf{b} \tag{3.17}$$

where $\mathbf{p}_i$ are the columns of $\mathbf{P}$, b_i are the initial amplitudes of each mode, $\mathbf{O}$ is a diagonal matrix containing the ω_i and $\mathbf{b}$ is the vector containing all the b_i, which can be calculated as follows:

$$\mathbf{x}(t=0) = \mathbf{x}_1 = \mathbf{P}\mathbf{b} \implies \mathbf{b} = \mathbf{P}^{\dagger}\mathbf{x}_1 \tag{3.18}$$

It is important to notice that $\mathbf{p}_i$ have the same dimension of $\mathbf{x}_k$, so the reconstructed field will have the same dimension as well.

Figure 3.1 illustrates the fundamental passages in the application of the algorithm to a simple fluid dynamics case.

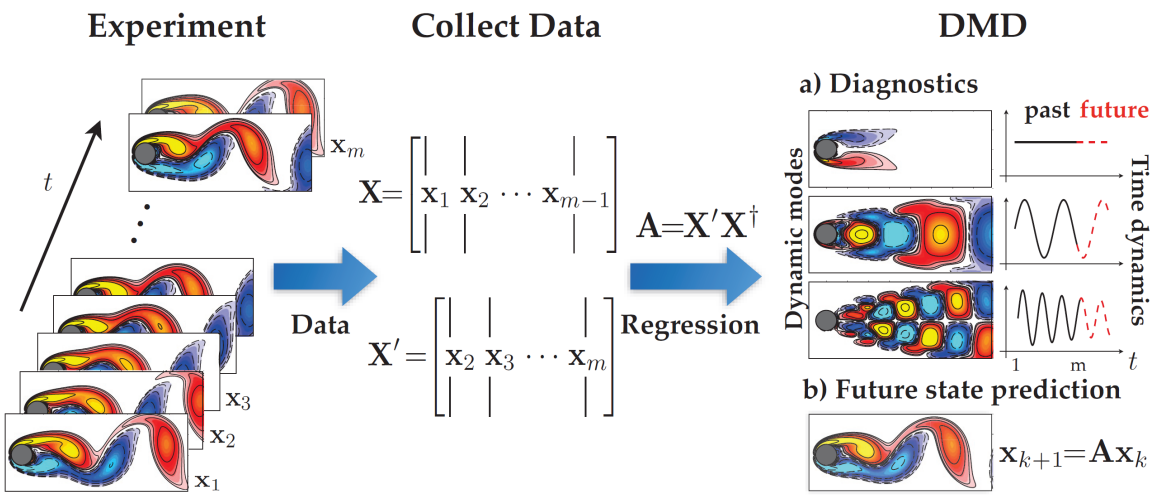


Figure 3.1. *Schematic overview of DMD, proposed by [Kutz et al., 2016], which shows the application of the algorithm to a fluid-dynamics case: the flow around a cylinder. The only difference here consists in the fact that one doesn't usually construct* $\boldsymbol{A}$*, but rather its projection onto POD modes* $\hat{\boldsymbol{A}}$ *in order to make the regression.*

Now that the algorithm has been clearly explained, we can proceed to show the Matlab implementation step by step, but beforehand we need to import OpenFOAM fields and save them into the Matlab Workspace. Inside the OpenFOAM case directory, a folder containing the desired field is present for each time instant in which data are saved, thus we need to enter in each one of them, read and store the field information, exit the folder and repeat. In order to accomplish our goal, the first step is the creation of a vector containing the names of every time folder, which, in case of a 0.01 time step, will be like $0.099975, 0.0200044, 0.0299986$ and so on, due to the adaptive time step option. After that, we can navigate all folders and gather the velocity field, which is commonly named `U` in OpenFOAM.

```
1 A = dir;    % struct array with files and directories names
2 T = 3.9;    % simulated time
3 dt = 0.01; % time step
4
5 for i=1:round(T/dt)
6         timeDir(i) = str2double(A(i+2).name);
7 end
8
```

```
timeDir = sort(timeDir); % time directory

X = []; % velocity field initialization

for i=1:length(timeDir)
        cd(sprintf('%.6f',timeDir(i)))
        fid = fopen('U');
        u = textscan(fid,'(%f %f %f)', 907200, 'HeaderLines', 22);
        X_up = [u{1};u{2};u{3}];
        X = [X X_up];
        fclose(fid);
        cd ..
end
```

Now we can apply the DMD algorithm step by step:

1. Firstly, the SVD decomposition and the reduction of matrices dimensions:

```
[U,S,V] = svd(X(:,1:end-1), 0);

r = 20; % must be less than length of timeDir

U = U(:,1:r);
S = S(1:r,1:r);
V = V(:,1:r);
```

   The second argument inside `svd` function deserves an explanation: it is needed to set the 'economy' version of SVD, which is the same as previously exposed. Without it, $\mathbf{U}$ would be $n \times n$ instead of $n \times m$, making it extremely hard to store. Also $\mathbf{S}$ would change from $m \times m$ to $n \times m$, but this would result in a minor issue.

2. Secondly, we will compute $\hat{\mathbf{A}}$:

```
Ahat = U' * X(:,2:end) * V / S;
```

3. Now we proceed with its eigendecomposition and the calculation of the continuous time eigenvalues

```
[W,L] = eig(Ahat);
lambda = diag(L);
omega = log(lambda) ./ dt;
```

4. We can now construct the eigenvectors of $\mathbf{A}$:

```
P = X(:,2:end) * V / S * W
```

5. Finally, the reconstruction of the velocity field, preceded by the calculation of $\mathbf{b}$:

```
b = P \ V(:,1);

for i=1:length(b)
    Pb(:,i) = P(:,i) .* b(i);
end

t = linspace(0,T-2*dt,length(timeDir)-1);

for i=1:length(t)
    Xdmd(:,i) = Pb * exp(omega .* t(i));
end
```

The vector `t` contains the time instants in which `Xdmd` will be calculated. We must clarify that `Xdmd` is the reconstructed version of `X(:,2:end)`, so the first snapshot will be excluded from the reconstruction. Another important aspect is that `Xdmd` will be a complex matrix, but clearly only the real part is relevant[7].

Now we need a way to compare our original field with the reconstructed one. The easiest method is to employ Paraview, the software used to visualize OpenFOAM simulations results. In order to do so, we must write down `Xdmd` as if it was an OpenFOAM field text file. We will thus have to enter every folder and create a text file containing the information of `Xdmd` at the corresponding time, then repeat till the end. Fortunately, Matlab allows us to easily write text files. It is important that the generated text files contain the minimum amount of code required to make OpenFOAM treat `Xdmd` as a `volVectorField` class object, the same as `U`.
Lastly, we want a way to measure the goodness of the DMD approximation. After the testing of different error formulations, the most significant one appears to be the following:

$$err(t) = \sqrt{\frac{\langle \mathbf{x}(t) - \mathbf{x}_{DMD}(t), \mathbf{x}(t) - \mathbf{x}_{DMD}(t) \rangle}{\langle \mathbf{x}(t), \mathbf{x}(t) \rangle}} \tag{3.19}$$

where $\mathbf{x}(t)$ and $\mathbf{x}_{DMD}(t)$ are the original and the reconstructed state vectors at a given time, while the brakets $\langle \bullet \rangle$ indicate the scalar product. We will thus have an error value for every time instant, so that it can be easily plotted directly in Matlab. Here is the corresponding code in the case of `X` and `Xdmd`:

```
Xdiff = X(:,2:end) - real(Xdmd);

for i=1:length(timeDir)-1
    err(i) = sqrt((Xdiff(i)' * Xdiff(i)) ./ (X(:,i+1)' * X(:,i+1)));
end
```

[7] It can be verified that the imaginary part is different orders of magnitude smaller than the real one.

3.3 Results

The presented algorithm was applied to two cases: both are LES simulations of a MHD flow across the backward facing step, one ending in a steady state and one keeping a turbulent regime. For both cases we will compare some of the original snapshots with the reconstructed ones and then plot $err(t)$ as reported in (3.19).

3.3.1 Steady state scenario

In this case, the steady state is reached in about $0.5 \div 0.6$ s and the simulated time is 1 s. The time step is of 0.02 s, thus bringing to a total of 50 snapshots. Since it is a small number, r will be 49, which is the largest possible[8]. Firstly, a visual comparison between the original field and the reconstructed one is reported in Figure 3.2.

(a) $t = 0.04$ s

(b) $t = 0.22$ s

(c) $t = 1$ s

Figure 3.2. *Comparison between the original and reconstructed field at the first reconstructed time step, in the middle of the transient and at the end of the simulation. For every time reported, the field above is the original one, while the other is the reconstruction.*

As we can see, the accuracy is higher in the middle and at the end. At the beginning, the method tends to overestimate the amplitude of some modes, which leads to the presence of nonexistent structures. The $err(t)$ plot can confirm what just asserted.

[8]This choice of r seems in contradiction to what we explained in 3.2, since now $r = m - 1$. The reason behind our choice is that calculations with such a small value of r are fast enough to allow the use of all snapshots, which naturally brings to more precise results.

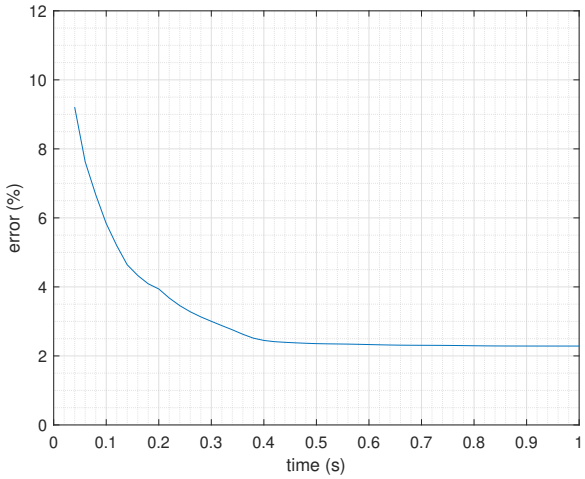


Figure 3.3. *Error plot vs time. As we expected, it is higher at the beginning, while it decreases to reach a constant value at the steady state.*

In conclusion, results are quite satisfying, considering that the maximum error is around 9%, but tends to decrease rapidly. Moreover, the recover of the steady state with only a 2.2% error is a confirmation of the correct implementation of the algorithm.

3.3.2 Turbulent scenario

In this second case, turbulence is stronger and the transient is longer (the simulated time is 3.9 s). Indeed, the previous, steady state example was more like a verification of the algorithm, while now we will analyze the impact of the time-sampling choice on the field reconstruction by testing three different time steps. Firstly, one of the most accurate results is reported and compared the original field (Figure 3.4).

(a) $t = 0.1$ s

(b) $t = 1$ s

(c) $t = 3.9$ s

Figure 3.4. *Comparison between original and reconstructed field at the first reconstructed time step, in the middle of the transient and at the end of the simulation. For every time reported, the field above is the original one, while the other is the reconstruction.*

As for the previous case, one can notice some small discrepancies at 0.1 s , which tends to vanish as time passes. The error will be now studied considering three different time steps: 0.1 s, 0.05 s and 0.01 s. For every one of them, we will try different values of r.

0.1 s

In this first case, the number of snapshots is 39, thus $r_{max} = 38$.

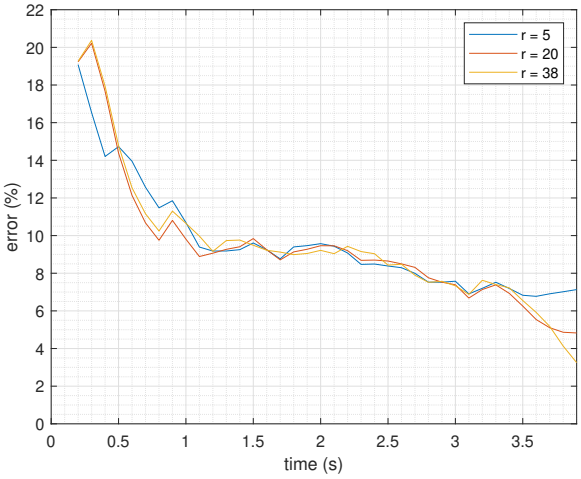


Figure 3.5. *Comparison between three different values of r, considering* 0.1 s *as time step.*

As we can see, the error is pretty high at the beginning, but decreases quickly in 1 second of simulated time. It is interesting to notice that the only sensible difference between the three cases is at the end of the simulation, where the largest r appears to be the most accurate.

0.05 s

In this second case, the number of snapshots is 78, thus $r_{max} = 77$.

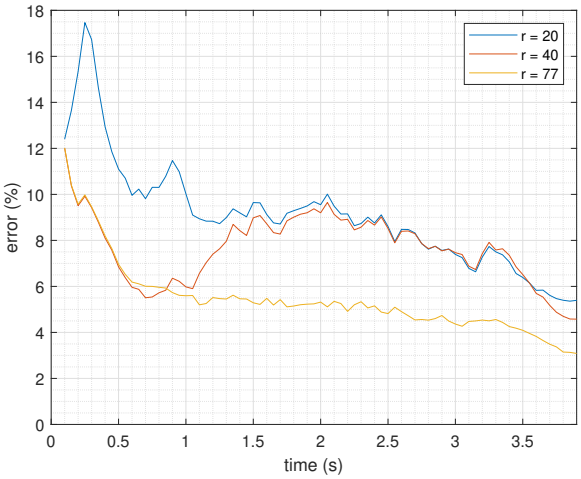


Figure 3.6. *Comparison between three different values of r, considering* 0.05 s *as time step.*

The difference between the three now is relevant across all the simulated time interval, with $r = r_{max}$ being the best choice by far.

0.01 s

In the last case the number of snapshots is 390, thus $r_{max} = 389$, which is pretty large. Indeed, calculations with such large r are extremely slow, so the rule $r < m$ should be applied, but, for the sake of completeness, we will test it out anyway.

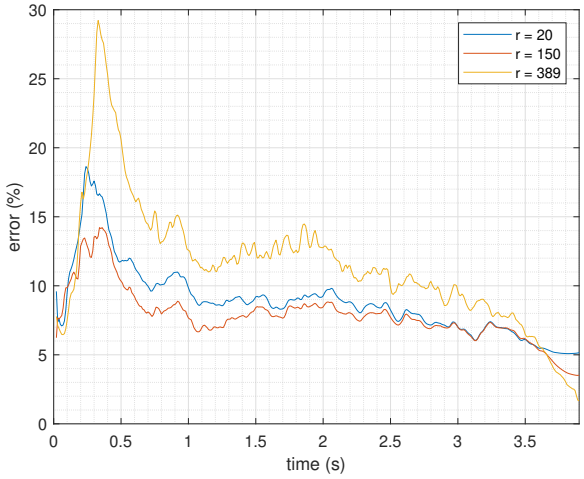


Figure 3.7. *Comparison between three different values of* r*, considering* 0.01 s *as time step.*

This last one is surely the most peculiar result: by using a too large value of r, the overestimation of modes in the beginning becomes more important, thus leading to imprecise results. On the other hand, the trend changes at the end, where $r = r_{max}$ becomes the most accurate option, making it look like the best choice for future predictions of the system. The problem resides in the fact that the computational time required for the reconstruction will be extremely high, making it less interesting.

3.4 Conclusions

in order to establish which is the best approach to follow in our case, we should make a comparison between the tested time steps. For each one of them, the value of r which brought to the most accurate result is considered. Figure 3.8 reports thus the errors obtained from the following combinations: 0.1 s with $r = 38$, 0.05 s with $r = 77$ and 0.01 s with $r = 150$.

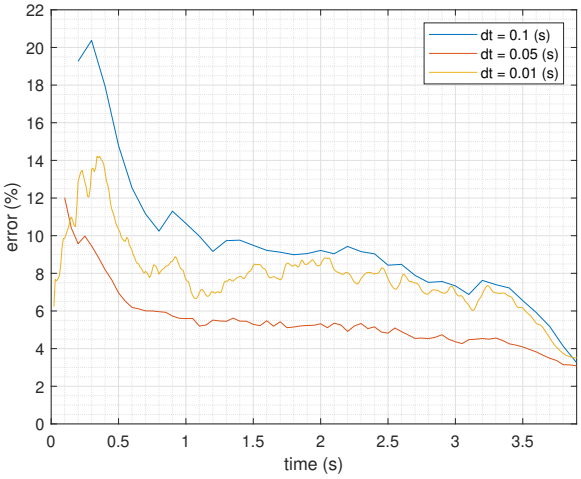


Figure 3.8. *Comparison between the lowest errors found for every time step.*

Figure 3.8 clearly shows that we obtained the most accurate reconstruction by employing a 0.05 s time step and the largest r possible, which in that case is 77, but it is worth mentioning that, close to the end, all of them tends to a similar value. For the sake of clarity, we tested both 0.05 s and 0.01 s with $r = 77$. The result is shown below.

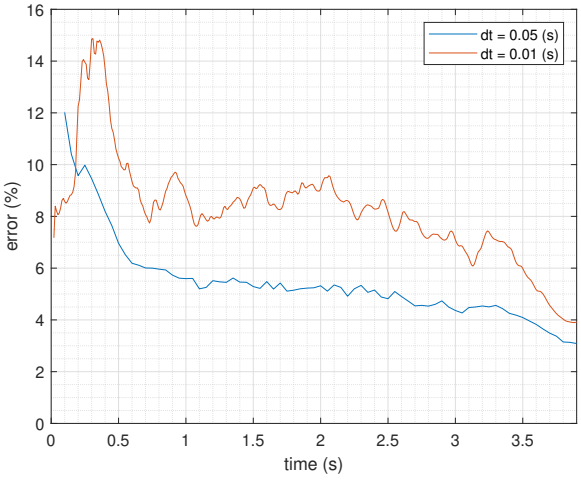


Figure 3.9. *Comparison between* 0.05 s *and* 0.01 s *employing* $r = 77$.

We can thus confirm that the best option is 0.05 s with $r = 77$. The fact that, by shortening the time step, the reconstruction diverges from the original result is quite peculiar. Concerning the physics of the system, one reason might be that a short time step led to an overestimation of the short time dynamics and underestimation of the

long time one. Another reason could reside in the numerical aspect of the algorithm: indeed, by employing a too short time step, the SVD decomposition of the matrix $\mathbf{X}_1^{m-1}$ could become more problematic[9], letting some numerical instabilities arise.
In conclusion, we are satisfied with our result: considering the strong non-linearity present in our physical system, the linear approximation provided by the DMD algorithm shows a good agreement with the original simulation. What makes it even more satisfying is the fact that the reconstruction takes less than 1 hour on a quad core laptop to be performed, while the full simulation takes several hours on a 15-core processor, thus making the DMD option viable for the reconstruction of the future state of an MHD system. Moreover, it opens to the possibility to realize both parametric and sensitivity analysis on LES simulations in an extremely reduced amount of time with respect to the traditional way, in which full simulations are repeated for every change in parameters. For sure these aspects could result in interesting future developments of the current work.

[9]Indeed, shorter time step means more columns in $\mathbf{X}_1^{m-1}$.

Conclusions

In this thesis work, a compressible MHD OpenFOAM solver has been developed and successfully verified on the Hartmann Flow case. In order to test its goodness in the application of turbulence models to MHD scenarios, it was then employed in the simulation of a conductive fluid passing through a backward facing step while under the effect of external magnetic field. Here, RAS and LES models were tested and the application of the latter brought to interesting results. Lastly, we applied the DMD algorithm to some of the LES simulations with good results, thus allowing for faster computational time.
After this brief resume, all the achievement reached throughout the three chapters of this work will be discussed and compared with the predetermined objectives. Some possible future developments are also addressed.

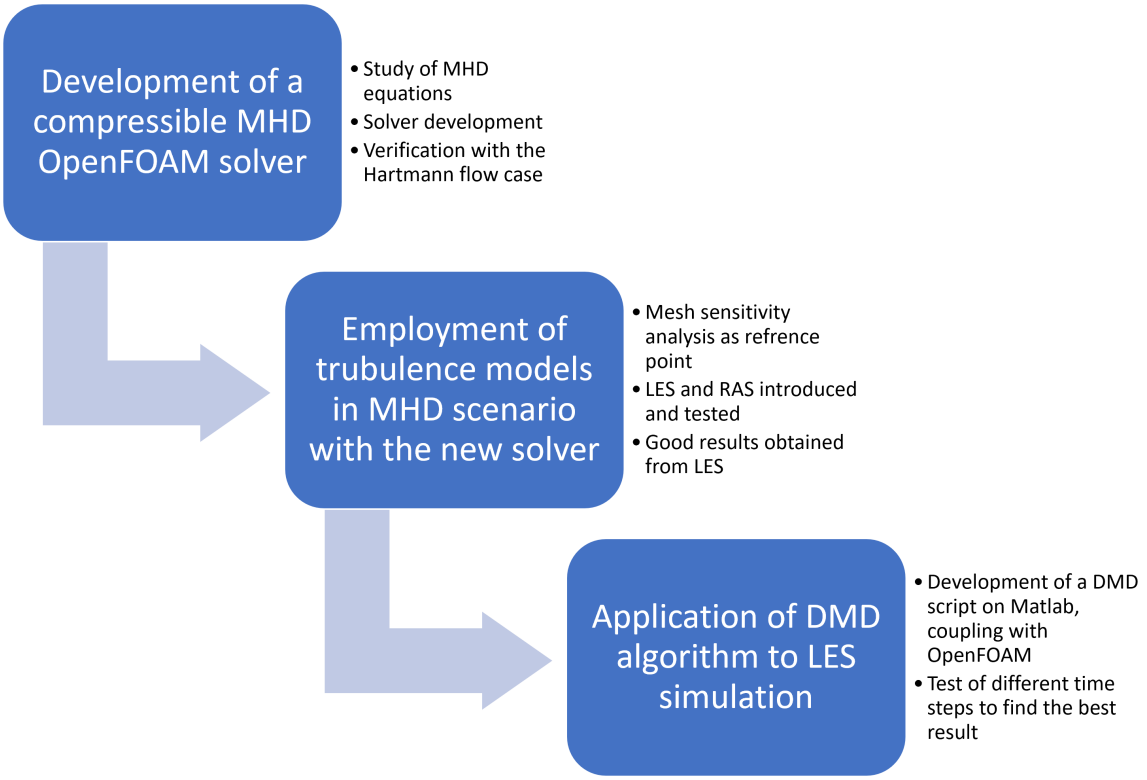
1. **Development and verification of a new MHD solver for OpenFOAM.** The solver comprMhdFoam was born as a tool to deal with compressible MHD, allowing also for the use of turbulence models if needed. The success of the verification on the Hartmann flow case was the proof that the physics we introduced was correctly implemented on the starting point solver, which only treats compressible fluid dynamics. It is important to remember that we wanted to exclude compressibility effects from the verification because the chosen starting point is well-known for its capability to simulate such scenarios.
   Starting from our accomplishments, one possible future developments of this chapter will be for sure the possibility to treat walls with arbitrary conductivity. As it is by now, the solver can only treat insulating walls, so a generalization in that sense would be extremely helpful. Moreover, the test or even a validation on a compressible MHD cases could be a better confirmation of its reliability.

2. **MHD flow across a backward facing step.** The objective in the second chapter was to verify the goodness of comprMhdFoam in employing turbulence models in an MHD case. They represent a way to obtain a nice compromise between accurate results and low computational time when studying turbulent phenomena and as such they are extremely helpful in engineering applications. The mesh sensitivity analysis was necessary to provide a reference point to evaluate the application of such models to the chosen problem. Results were surely satisfying when employing LES and for this reason we were encouraged to explore some of the features of our physical system, such as three-dimensional effects and reattachment length.
   Many complex cases can be investigated to test the capability of the new solver. One example is the turbulent motion which can arise in presence of heat transfer:

the effect of an external magnetic field on such situation is for sure interesting to observe ([Belyaev et al., 2019] studied the phenomenon through an experiment). Another interesting aspects are the pressure drops due to the Lorentz force, maybe coupled with a complicated geometry.

3. **Dynamic Mode Decomposition approach.** The DMD algorithm was applied to some LES simulation in order to test the capability of the method to provide an efficient way to save computational time. Results were reported under various conditions, related to different time-sampling and number of modes kept for the reconstruction, and proved to be satisfyingly accurate. For this reason, in future works related to the presented topics, the DMD algorithm could become the key to save a large amount of computational time.

A flux diagram which summarizes all the passages of the presented work is reported in the following.



Appendix A

Linear stability analysis of two-dimensional simulations

In this appendix, a tool for the *linear stability analysis* is presented and tested on some two-dimensional OpenFOAM simulations carried out during Chapter 2. It is based on the linearization of the underlying equations of the physical system and the description of the perturbations[1] with a Fourier-like analysis. Such tool was developed by [Trotta, 2019] in Matlab, thus allowing its application also to OpenFOAM profiles[2]. The algorithm is reported step-by-step:

1. The profiles of u_x, u_y, T and ρ, taken at a given time instant along a section parallel to y of the geometry, are interpolated on Chebyshev nodes. After some testing, it appears that 300 nodes are enough for an accurate interpolation.

2. The problem is linearized, considering the following formula for perturbations:

$$\delta f(y) e^{i(\xi x - wt)} \tag{A.2}$$

   where $\delta f(y)$ represents the perturbation amplitude of a generic quantity, ξ is the wave number (m^{-1}) and w is the complex time frequency (s^{-1}). Then, the eigenvalues of the resulting algebraic system, i. e., the aforementioned w, are studied for every value of ξ specified by the user.

3. Since the stability of our system is associated to the condition $Im(w) < 0$[3], for every specified ξ the eigenvalue with the maximum imaginary part is selected and compared to such value.

4. The process is repeated for every time instant in which data fields are saved and the aforementioned imaginary part plotted against time.

[1] When performing a linear stability analysis, each variable of the differential problem is written as:

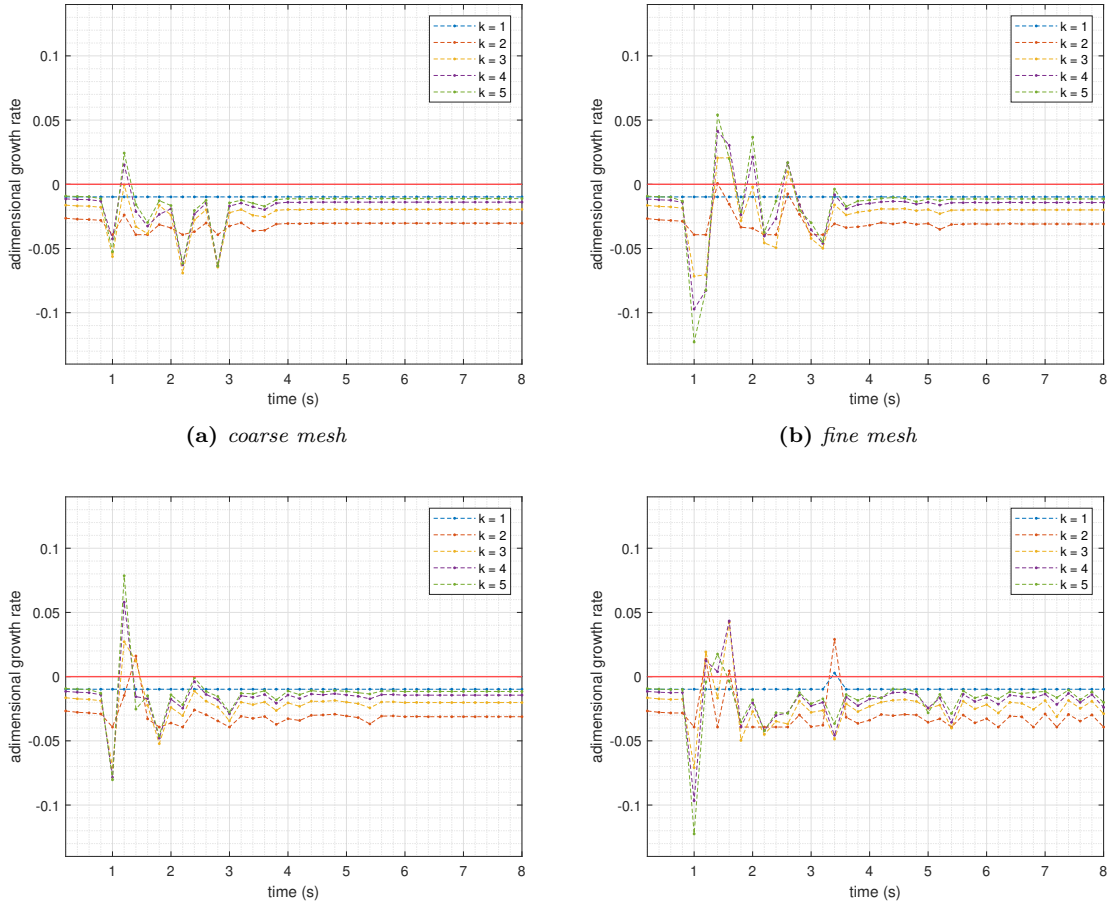
$$g(\vec{x}, t) = g_0(\vec{x}) + \delta g(\vec{x}, t) \tag{A.1}$$

where g_0 represent the steady state value of the generic variable g and δg represent a small perturbation.

[2] The custom-made `readFields` function, employed in Chapters 1 and 2, will be used also in the present case.

[3] Indeed, if such condition is not respected, a perturbation of the form (A.2) would increase in value indefinitely, thus bringing to an unstable situation.

Firstly, we will apply the algorithm to the profiles obtained from the simulations carried out during the mesh sensitivity analysis (Subsection 2.2.3). The graphs reported in Figure A.1 shows the plots of the maximum imaginary parts for a given set of ξ values versus time. These values were chosen as the most significant ones because they brought to the most unstable behavior. In all simulations, $Re = 10000$, $Ha = 24$ and the profiles are taken 27 cm away from the step.

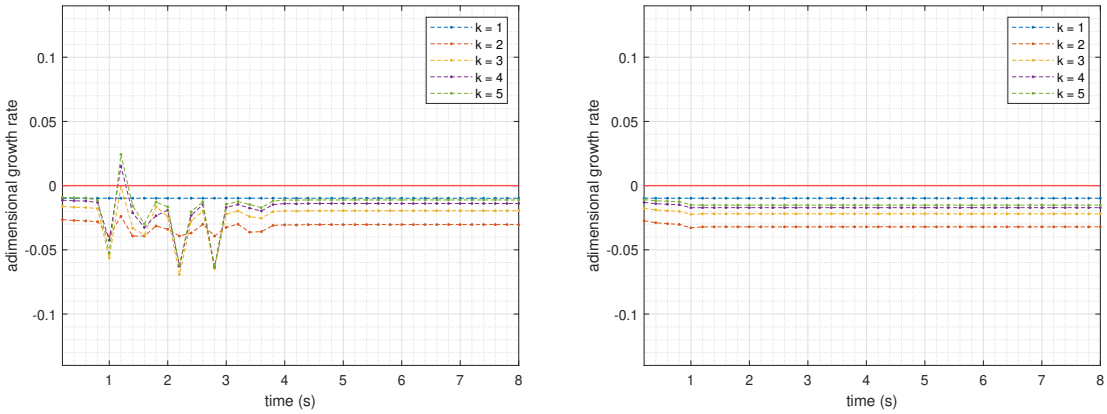


(a) *coarse mesh* **(b)** *fine mesh*

(c) *finer mesh* **(d)** *finest mesh*

Figure A.1. *Comparison between the stability analysis of simulations performed on the four meshes.*

The flatness of the plots corresponds to a steady state in the stability analysis, which corresponds to the steady state found during the simulations. Its absence in the case of the finest mesh is also confirmed.
The tool was also applied to the RAS simulation reported in Subsection 2.3.2 and compared to the image (a) in Figure A.1, since both were performed on the same mesh. Figure A.2 shows the comparison. $Re = 10000$, $Ha = 24$ and the profiles are taken 27 cm away from the step also in this case



(a) *no model* **(b)** *RAS*

Figure A.2. *Stability analysis comparison: coarse mesh without model and with RAS model.*

The result is coherent with the nature of the RAS approach: since all the fluctuations are cut out from the solution, the system behaves in a more stable way and the stability plot is flat almost from the start.

Bibliography

[Belyaev et al., 2019] Belyaev, I. A., Biryukov, D. A., Kotlyar, A. V., Belavina, E. A., Sardov, P. A., and Sviridov, V. G. (2019). Heat Transfer during Mixed Convection of a Molten Salt in the Presence of Magnetic Fields. *Technical Physics Letters*, 45(5):499–502.

[Bodi et al., 2018] Bodi, S. U., Bokade, V., Bhandarkar, U., Gopalakrishnan, S., and Kowsik (2018). Numerical simulation of two-fluid magnetohydrodynamic channel flows. 74:1342–1352.

[Chen and Chen, 2018] Chen, F. F. and Chen, F. F. (2018). *Erratum to: Introduction to Plasma Physics and Controlled Fusion*.

[Di Ronco et al., 2020] Di Ronco, A., Introini, C., Cervi, E., Lorenzi, S., Jeong, Y. S., Seo, S. B., Bang, I. C., Giacobbo, F., and Cammi, A. (2020). Dynamic mode decomposition for the stability analysis of the Molten Salt Fast Reactor core. *Nuclear Engineering and Design*, 362(January):110529.

[Dousset, 2009] Dousset, V. (2009). Numerical simulations of MHD flows past obstacles in a duct under externally applied magnetic field. (December):1–183.

[Ferroni, 2012] Ferroni, F. (2012). Magneto-hydrodynamic Simulations of Liquid Metal Flows in Fusion Reactors. (June):58.

[Freidberg, 2007] Freidberg, J. P. (2007). *Plasma Physics and Fusion Energy*. Cambridge University Press, Cambridge.

[Kutz et al., 2016] Kutz, J. N., Brunton, S. L., Brunton, B. W., and Proctor, J. L. (2016). *Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems*.

[Mas de Les Valls et al., 2012] Mas de Les Valls, E., Batet, L., Medina, V., and Sedano, L. (2012). MHD thermofluid flow simulation of channels with a uniform thermal load as applied to HCLL breeding blankets for fusion technology. *Magnetohydrodynamics*, 157-168:157–168.

[Paudel et al., 2019] Paudel, S., Bhattarai, S., Bhattrai, S., and Bomjan, B. (2019). Effects of magnetohydrodynamics on temperature and shock standoff distance in a supersonic flow over a blunt body. *Progress in Computational Fluid Dynamics*, 19(4):264–271.

[Ryakhovskiy and Schmidt, 2016] Ryakhovskiy, A. and Schmidt, A. (2016). MHD supersonic flow control: OpenFOAM simulation. *Proceedings of the Institute for System Programming of the RAS*, 28(1):197–206.

[Sobolev, 2010] Sobolev, V. (2010). Database of thermophysical properties of liquid metal coolants for GEN-IV. *SCK• CEN Technical Report, SCKCEN-BLG-1069*, 16(12):3496–3502.

[Tano-Retamales et al., 2019] Tano-Retamales, M., Rubiolo, P., and Doche, O. (2019). Development of Data-Driven Turbulence Models in OpenFOAM: Application to Liquid Fuel Nuclear Reactors. *OpenFOAM®*, pages 93–108.

[Tassone, 2016] Tassone, A. (2016). CFD with OpenSource software Magnetic induction and electric potential solvers for incompressible MHD flows Learning outcomes.

[Trotta, 2019] Trotta, G. (2019). An innovative approach for stability analysis of conductive fluids : from incompressible to compressible formulation.

[Woelck and Brenner, 2017] Woelck, J. and Brenner, G. (2017). Large-eddy-simulation of turbulent magnetohydrodynamic flows. *Thermal Science*, 21:S617–S628.