CHALMERS

Simulering av kompressibelt flöde i Python, MMSX21

Utveckling av numeriska metoder för hantering av stötar i kompressibelt flöde

ARVID ANGEL
MELVIN JOHANSSON
LUKAS LARSSON
JESPER WISTRÖM

INSTITUTIONEN FÖR MEKANIK OCH MARITIMA VETENSKAPER

CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2023
www.chalmers.se

Kandidatarbete VT 2023

Simulering av kompressibelt flöde i Python MMSX21

Utveckling av numeriska metoder för hantering av stötar i kompressibelt flöde

Kandidarbete i mekanik och maritima vetenskaper

ARVID ANGEL
MELVIN JOHANSSON
LUKAS LARSSON
JESPER WISTRÖM

CHALMERS

Institutionen för mekanik och maritima vetenskaper
Avdelningen för strömningsmekanik
Chalmers Tekniska Högskola
Göteborg, Sverige 2023

Simulering av komprimerbart flöde i Python MMSX21
Utveckling av numeriska metoder för hantering av stötar i kompressibelt flöde
ARVID ANGEL
MELVIN JOHANSSON
LUKAS LARSSON
JESPER WISTRÖM

Handledare: Debarshee Gosh,
Institutionen för mekanik och maritima vetenskaper
Examinator: Niklas Andesson,
Institutionen för mekanik och maritima vetenskaper

Kandidatuppsats 2023
Institutionen för mekanik och maritima vetenskaper
Avdelningen för strömningsmekanik
Chalmers Tekniska Högskola
SE-412 96 Göteborg
Telephone +46 (0)31 772 1000

Omslag: Mach-tal visualiserat i höghastighetsflöde över en ramp

Institutionen för mekanik och maritima vetenskaper Typeset in LaTeX

Göteborg, Sverige 2023

Abstract

Computational fluid dynamics (CFD) simulations are an essential part of product development. If the simulations are physically correct, it can reduce the costs of experiments that otherwise would have to be done to evaluate the interactions between the product and the fluid it flows in or the fluid that flows past a stationary product. When the fluid velocity equals to or exceeds the speed of sound, it introduces a phenomenon called *shocks*.

The project aimed to investigate whether replacing the existing shock sensor in a CFD solver, called "G3Dflow", could increase the accuracy and precision of the simulations. "G3Dflow" is a Python library provided by the Institution of Mechanics and Maritime Sciences at Chalmers.

The process of choosing a new shock sensor was mainly a matter of studying literature. One part of the project focused on how to calculate the gradients needed to compute the new sensor and how this affected the simulations. A major part of the project was evaluating the results of the simulations and comparing the results to literature describing test cases and the expected outcome. The results of the project are then to be used as a case for the course TME085-*COMPRESSIBLE FLOW*.

Förord

Rapporten presenterar resultaten av en kandidatuppsats utförd vid Institutionen för mekanik och maritima vetenskaper på Chalmers tekniska högskola i Göteborg, Sverige. Målet med projektet var att implementera känsligare sensorer för stötdämpning i supersoniskt flöde i Chalmers egna CFD-kod ”G3DFlow.py“ och på så sätt förbättra koden för viskösa fall. Ytterligare ett mål är att konstruera ett Case, en simulerings-uppgift med komplementerande uppgifter samt facit till dessa, som skall användas i kursen TME085-*COMPRESSIBLE FLOW*.

Projektet genomfördes under perioden 27 Januari-10 Maj 2023. Det rekommenderas att läsaren av denna uppsats har grundläggande kunskaper om strömningsmekanikens principer.

Tack

Vi vill tacka Professor Niklas Andersson för möjligheten att arbeta med detta intressanta projekt och för hans hjälpsamma vägledning under hela arbetet. Ett stor tack också till våra handledare, doktoranden Debarshee Ghosh, för hans hjälp genom alla steg av projektet, samt till Institutionen som tillhandahöll koden.

Nomenklatur

Nedan följer nomenklatur för index, parametrar och variabler som används genom rapporten.

Index, parametrar och variabler

i,j	Index för cellnummer
t	Tid
Δt	Tidssteg
ρ	Densitet
V	Volym
S	Yta
A	Area
$d\mathcal{V}$	En oändligt liten volym
$d\mathcal{S}$	En oändligt liten yta
$\mathbf{V}$	Lokal hastighet
$\mathbf{n}$	Normal
$\mathbf{F}$	Kraft
$\mathbf{f}$	Kraft per massenhet
p	Tryck
W	Arbete
e	Inre energi
u	Lokal hastighet
h	Entalpi
Q	Värme
q	Specifikt värme
Ψ	Jameson-sensor-koefficient
Φ	Ducros-sensor-koefficient

∇	Gradientoperator
ϵ	Ett mycket litet tal
ω	Vorticitet
g	gravitation
Ω	Område/kropp/volym
$fnctp$	Faktor mellan 0-1 som anger till vilken grad stötarna ska dämpas
$cfilt$	Beräknad dämpningskoefficient

Innehåll

Figurer

Tabeller

1
Inledning

Denna sektion är en introduktion till strömningsmekanik, utvecklingen av CFD och hur det kan tillämpas till kompressibla flöden. Syftet med projektet samt projektets problembeskrivning och avgränsningar beskrivs för att ge en överblick över hur projektet grundades, i vilken riktning som gruppen arbetade och varför arbetet är relevant.

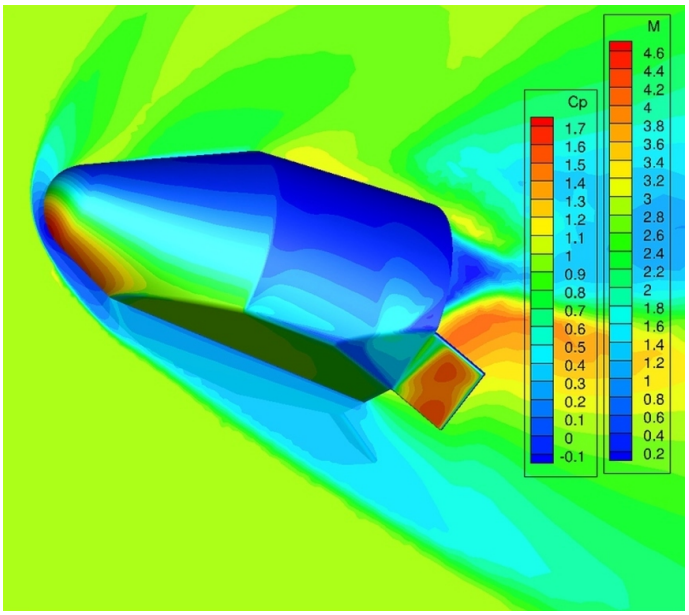
1.1 Bakgrund

Strömningsmekaniken grundar sig i tre styrande ekvationer: bevarande av massa, bevarande av rörelsemängd och bevarande av energi. Med dessa, så kallade Navier-Stokes ekvationer [5, s.46, s.47, s.52], kan exempelvis flödets tryck, temperatur, densitet, hastighet, entropi, skjuvning, vorticitet och mycket mer beräknas. Dessa erhållna flödesegenskaper kan sedan användas för att räkna på flödets inverkan på en kropp eller yta. Strömningsmekanik beskriver fluider, alltså kroppar som rör sig genom gaser (exempelvis luft) och vätskor (exempelvis vatten) samt sådant som tillämpar flöden till drivmedel, hydralik och mycket mer [8, s. 3]. Med andra ord omfattar strömningsmekaniken ett mycket stort område och är högst relevant för ingenjörsapplikationer. För att citera Frank M. White: "*Nästan allt på denna planet är antingen en fluid eller rör sig genom eller nära en fluid. Eftersom 75 procent av Jorden täcks av vatten och 100 procent täcks av luft är omfattningen av fluidmekanik enorm och berör nästan allt mänsklig bemödande*"[8, s. 5, egen översättning].

1.1.1 CFD och G3D::Flow

Sedan strömningsmekanikens begynnelse, med Newtons *Principia* 1687, fram till mitten av 1960 talet har teori tillsammans med experimentell data använts för att lösa förenklade och reducerade flödesproblem [5, s. 428]. Under 60-talet utvecklades CFD för att utföra mer komplicerade beräkningar och till stor del i militära syften då det blev relevant att simulera trubbiga raketspetsar (till rymdraketer och interkontinentala missiler). Med CFD kunde de tre styrande Navier-Stokes ekvationerna lösas numeriskt även för mer komplicerade geometrier och flöden, vilket dessutom var mindre kostsamt än att utföra fysiska experiment, [7]. Sedan dess har CFD blivit mycket mer sofistikerat, tillgängligt och ur ett modernt kontext nästan oundvikligt vad gäller strömningsmekanik. Idag har CFD blivit den tredje pusselbiten för att lösa flödesproblem som komplement till teori och expe-

rimentell data, och används i hög grad för designteknik för till exempel fordon. Figur 1.1 visar *Blue Origin* raketen 1.1 som exempel på vad som kan simuleras med hjälp av CFD, notera den trubbiga nosen. Speciellt relevant för detta projekt är CFD:s tillämpningar till kompressibelt flöde och Chalmers har till detta syfte under de senaste 10 åren utvecklat "G3D::Flow". Kompressibelt flöde är ett område inom strömningsmekaniken som beskriver flöden med mycket snabba hastigheter, ofta flera hundra meter per sekund, mer om detta i kapitel 2. G3D::Flow, som framförallt har använts i forskningsprojekt, är en intern kod som grundar sig i C++ men som nu har skrivits om till ett Python-bibliotek, "G3DFlow.py". Python-biblioteket är specialiserat för att lösa fall med kompressibelt flöde och ska i nuläget korrigeras, testas och valideras innan den integreras tillbaka in i det mer omfattande ursprungsbiblioteket "G3D::Flow".



Figur 1.1: CFD-simulering av Blue Origin rymdraket [12]

1.2 Syfte

Det huvudsakliga syftet med projektet är att förbättra python-bibliotekets stöthantering. Arbetets syfte inkluderar även att testa och kartlägga förbättringarna mot kända testfall i både icke-viskösa och viskösa fall. Sist skall ett projekt-case konstrueras till kursen *COMPRESSIBLE FLOW*-TME085 med hjälp av det förbättrade python-biblioteket.

1.3 Problemformulering och Frågeställning

I detta avsnitt beskrivs kort de problem som åtgärdades och varför just dessa problem bearbetades. Eftersom problemlösningen bygger på förståelse av en mängd teori beskrivs teorin mer utförligt i kapitel två medan tillvägagångssättet för problemlösningen beskrivs i kapitel tre. Som nämnt i bakgrunden är CFD ett mycket

omfattande område och python-biblioteket "G3DFlow.py" är en mycket omfattande kod även om den framförallt berör kompressibelt flöde. Under projektets gång kommer gruppen därför arbeta utifrån hänvisning av handledare och examinator för att lösa de mest relevanta problem som berör koden i nuläget.

En ny stötsensor kommer att implementeras för att göra simuleringen bättre på att identifiera stötar och skilja dem från stora gradienter orsakade av till exempel turbulens. Tidigare användes den så kallade Jameson-sensorn som beskrivs i 2.5.1. En ny implementation kommer att åtgärda Jameson-sensorns tillkortakommanden utan att försämra den existerande koden. Resultaten kommer sedan jämföras med existerande litteratur och experimentell data för att validera att den nya implementationen är korrekt.

Med denna problemformuleringen ställdes följande frågor:

- Hur kommer implementationen av nya verktyg i koden påverka simuleringens förmåga att lösa standardflödesfall?
- Hur kommer den nya kodens implementation jämföra sig med experimentell data och hur realistisk kommer den nya koden bete sig?
- Hur kommer implementationen av nya stötvågssensorer påverka förmågan att identifiera stötar?
- Hur kommer behandlingen av viskösa gränsskikt påverka förmågan att prediktera flödesfall?
- Hur kommer den nya implementationen jämföra sig med originalkoden vad gäller, till exempel, noggrannhet och beräkningskostnad?

1.4 Avgränsningar

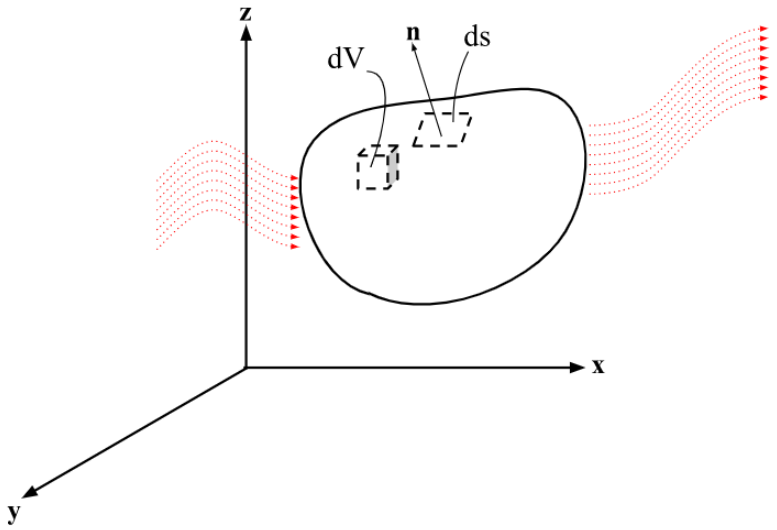
För att begränsa projektets omfattning görs en del avgränsningar. Till att börja med berör projektet endast komprissibelt flöde för en uppsättning standardtestfall. Anledningen till denna avgränsning är att fokuset för kandidatarbetet ligger på att öka kodens funktionalitet, vilket förhoppningsvis kommer att kunna tydas ur testfallen. Att använda fler testfall hade tagit mycket tid och energi eftersom de måste konfigureras och anpassas för att kunna användas med koden. Antalet testfall anses vara tillfredsställande och tillräckligt för att avgöra om koden förbättrats. Resultat av simuleringar kommer att ytligt jämföras med litteratur som beskriver förväntat utfall, för att säkerställa att resultat är verklighetstrogna och rimliga. Tillsammans med antagandet att det ursprungliga biblioteket simulerar icke-viskösa flöden med hög noggrannhet utöver stöthanteringen bedöms det vara tillräckligt för att kunna fastställa huruvida den nya sensorn förbättrar resultaten eller ej. Observera att biblioteket stödjer implementation av turbulens, men att turbulens inte förväntas förekomma i något av testfallen. Projektet kommer inte heller att behandla turbulens, vi avgränsar oss till att endast använda ett laminärt testfall i det viskösa flödesfallet.

2
Teori

Följande kapitlet behandlar de styrande Euler ekvationerna som CFD mjukvaran faktiskt löser numeriskt, men också de teoretiska grunderna inom kompressibelt flöde, som stötar och även metoderna för att ta hand om stötarna när flöden simuleras. Utöver teorin som rör flödesdynamiken förklaras även teorin som ligger bakom den numeriska lösningen av flödesproblem, i synnerhet under förhållanden som ger upphov till kompressibelt flöde. I denna teori del kommer det förklaras relativt omfattande hur simuleringen fungerar, vilka problem som uppstår på grund av stötar och vilken teori som tillämpas för att åtgärda dessa problem.

2.1 Eulers ekvationer

Eulers tre konserveringsprinciper är de i grunden styrande ekvationerna. Som nämnt i bakgrunden är dessa helt enkelt bevarande av massa, röreslemängd och energi då en fluid strömmar genom en kontrollvolym, se figur 2.1. Med dessa ekvationer kan flödets olika egenskaper analyseras, till exempel kan flödets tryck, hastighet, temperatur, densitet, etc, beräknas i olika punkter i flödet. Navier-Stokes ekvationerna är mycket lika Eulerekvationerna i och med att de också bevarar massa, rörelsemängd och energi över en kontrollvolym men utöver det också tar hänsyn till något som kallas *viskositet*, se kap 2.2. Eulers ekvationer berör inte viskositet, men som skrivs om i 2.3 är det viskösa gränsskiktet mycket tunt för flöden med höga hastigheter, vilket är exakt det som studeras i detta projekt som omfattar kompressibelt flöde. Därför kan det antas att Eulers ekvationer ger en bra representation av de fysikaliska processer som styr flödet och dessa beskrivs i denna sektion.



Figur 2.1: Flöde genom en kontrollvolym

2.1.1 Kontinuitet

I figur 2.1 används en så kallad kontrollvolym för att härleda Eulerekvationerna. Kontrollvolymen har en total volym V och area S. Låt nu en oändligt liten volym $d\mathcal{V}$ multipliceras med en lokal densitet ρ för att få massan inuti denna "cell". Det antas att om en tillräckligt liten cell används kommer densiteten vara konstant. Densiteten integreras nu över hela kontrollvolymen för att få den totala massan inuti volymen. Om denna massan nu ändras över tiden, det vill säga om flödet är instationärt, tas derivatan med avseende på tid för att få förändring av massa inuti volymen:

$$\frac{d}{dt} \oiiint_{\mathcal{V}} \rho d\mathcal{V}$$

Över en oändligt liten yta dS kan det flöda både in och ut ur kontrollvolymen. Här måste därför hänsyn tas till ytans normal **n** och det noteras att ytan skrivs om på notationen $d\mathcal{S} = dS \cdot \mathbf{n}$. Över ytan $d\mathcal{S}$ strömmar ett flöde med lokala hastigheten **V** vars skalärprodukt med $d\mathcal{S}$ ger volymen per sekund som strömmar in eller ut ur kontroll volymen. Multiplicera detta med densiteten ρ och integrera över hela kroppens yta S fås nu nu massa per sekund som strömmar genom kontrollvolymens yta:

$$\oiint_{S} \rho V \cdot d\mathcal{S}$$

Sammanställs dessa uttryck kan det ses att summan av förändringen av massa inuti en kontroll volym och massan som strömmar in eller ut ur en kontrollvolym måste vara noll eftersom massa varken kan skapas eller förstöras, det vill säga, kontinuitet:

$$\frac{d}{dt}\iiint_{\mathcal{V}} \rho d\mathcal{V} + \iint_{S} \rho V \cdot d\mathcal{S} = 0 \tag{2.1}$$

2.1.2 Konservering av rörelsemängd

Rörelsemängd beräknas enligt Newtons andra lag och anger att förändringen av en kropps rörelsemängd är lika med nettokraften som utövas på en kropp:

$$\frac{d}{dt}(m\mathbf{V}) = \mathbf{F} \tag{2.2}$$

Kraften **F** i högerled av 2.2 ges genom att igen ta en oändligt liten volym $d\mathcal{V}$, multiplicera med en kroppskraft **f** (som gravitation) och en lokal densitet ρ och sedan integrera över hela volymen. Denna kroppskraft är dock inte den enda kraften som verkar på kontrollvolymen då fluiden inuti kroppen kommer ha ett tryck som verkar utåt (till exempel för ett trycksatt kärl). Det antas igen att trycket över en tillräckligt liten yta dS kommer vara konstant. Multipliceras en mycket liten yta $d\mathcal{S}$ med trycket som verkar på denna yta fås en kraft som alltid kommer peka ut ur kontrollvolymen. Trycket i sig har ingen riktning, men i och med att trycket alltid ger upphov till en kraft som pekar ut ur kontrollvolymen måste kraftbidraget subtraheras (då kraft vektorn pekar i motsatt riktning relativt ytans normal). Sedan integreras denna över hela ytan för att summera kraftbidraget från alla små ytor $d\mathcal{S}$. Sammanställs kroppskraften och trycket över kroppens yta ges nu:

$$\mathbf{F} = \iiint_{\mathcal{V}} \rho \mathbf{f} d\mathcal{V} - \iint_{S} p d\mathcal{S} \tag{2.3}$$

För att framställa vänsterledet $\frac{d}{dt}(m\mathbf{V})$ i 2.2 används samma logik som för kontinuitets ekvationen. Rörelsemängd ges av produkten mellan massa och hastighet. För en liten volym $d\mathcal{V}$ fås massan genom att multiplicera med den lokala densiteten ρ. Sedan multipliceras detta med den lokala hastighetsvektorn **V** för att få rörelsemängden i en mycket liten"cell". Bidraget från alla celler summeras genom att integrera över hela kontrollvolymen. Slutligen ges förändringen av rörelsemängden genom att derivera med avseende på tid:

$$\frac{d}{dt}\iiint_{\mathcal{V}} (\rho \mathbf{V}) d\mathcal{V}$$

Slutligen strömmar massa in och ut ur kontrollvolymen och denna massa medför förändringar av rörelsemängden. För en liten yta $d\mathcal{S}$ ges massflödet av $\rho \mathbf{V} d\mathcal{S}$. Multiplicerat med mass flöde och med den lokala hastighetsvektorn **V** fås ett sorts flöde av rörelsemängd. Integrerat över hela kroppens yta ger summan av alla bidrag av rörelsemängdflödet:

$$\iint_{S} ((\rho \mathbf{V}) \cdot d\mathbf{S})\mathbf{V}$$

Sammanställs alla dessa uttryck fås ekvationen för förändringen av rörelsemängd:

$$\frac{d}{dt}(m\mathbf{V}) = \frac{d}{dt}\iiint_{\mathcal{V}}(\rho\mathbf{V})d\mathcal{V} + \oiint_{S}((\rho\mathbf{V})\cdot d\mathbf{S})\mathbf{V} \tag{2.4}$$

Sätt nu ihop höger och vänsterled 2.3 respektive 2.4 för att slutföra härledningen av 2.2:

$$\frac{d}{dt}\iiint_{\mathcal{V}}(\rho\mathbf{V})d\mathcal{V} + \oiint_{S}((\rho\mathbf{V})\cdot d\mathbf{S})\mathbf{V} = \iiint_{\mathcal{V}}\rho\mathbf{f}d\mathcal{V} - \oiint_{\mathcal{S}}pd\mathcal{S} \tag{2.5}$$

2.1.3 Energiprincipen

För konserveringen av energi gäller på liknande sätt som för masskontinuitet att energi varken kan skapas eller förstöras och därför är energin som förändras inuti kontroll volymen lika med energin som flödar in i (eller ut ur) kontroll volymen. Med andra ord, energin som tilläggs till en kontrollvolym per sekund (värme, $\dot{Q}$) plus energin som utförs av fluiden per sekund (effekt, $\frac{dW}{dt}$) är lika med förändringen av energin i fluiden (inre energi I.E plus kinetisk energi K.E):

$$\dot{Q} + \frac{dW}{dt} = \frac{d}{dt}(I.E + K.E) \tag{2.6}$$

Energin som flödar in i en mycket liten volym $d\mathcal{V}$ i form av värme ges av notationen $\dot{q}$ som har enheten $[\frac{J}{kg*s}]$. Denna volym är så pass liten att det antas att $\dot{q}$ är konstant i alla punkter av cellen. För att få fram enheten $[\frac{J}{s}]$ är det nu rimligt att multiplicera den specifika (per kg) effekten med massan av cellen, vilket ges av produkten mellan volymen $d\mathcal{V}$ och lokala densiteten ρ. I cellen tillförs nu värmen $\dot{q}\rho d\mathcal{V}$ och för att få totala värme tillförseln, det vill säga bidraget från alla celler, måste uttrycket integreras:

$$\dot{Q} = \iiint_{\mathcal{V}}(\rho\dot{q})d\mathcal{V} \tag{2.7}$$

Arbete per sekund är förstås en effekt som ges av $\frac{dW}{dt} = \mathbf{FV}$ där F är en kraft som utförs av fluiden som strömmar med en lokal hastighet V. Som för rörelsemängdshärledningen finns två befintliga krafter nämligen kraften som orsakas av fluidens tryck p över en yta $d\mathcal{S}$, och kroppskraften $\mathbf{f}$ över en volym $d\mathcal{V}$. Dessa summeras för att få nettokraften mellan tryck och kroppskraft och sedan integreras dessa över hela domänen för att få effekten som fluiden utövar på en kropp då den strömmar genom en kontrollvolym:

$$\frac{dW}{dt} = \mathbf{FV} = \iiint_{\mathcal{V}}\rho(\mathbf{f}\cdot\mathbf{V})\cdot d\mathcal{V} - \oiint_{S}\rho\mathbf{V}\cdot d\mathbf{S}. \tag{2.8}$$

Förändringen av energin i fluiden kan kan igen ses som förändringen inuti *volymen* plus energin som strömmar över *ytan*. Om $e + \frac{\mathbf{V}^2}{2}$ är specifika energin av en fluid bör det vid detta läge inte vara förvånande att multiplicera med massan av en liten volym, $\rho d\mathcal{V}$, integrera över hela kontrollvolymen och slutligen ta derivatan med avseende på tid. Sedan ges energin som strömmar genom kontrollvolymen genom att multiplicera den specifika energin med massflödet över en liten yta ($\rho \mathbf{V} d\mathbf{S}$) (enhetskontroll: $[\frac{kg}{m^3} * \frac{m}{s} * m^2] = [\frac{kg}{s}]$). Integrera nu över kroppens hela yta för att få med alla cell bidrag. Dessa termer adderas för att få:

$$\frac{d}{dt}(I.E + K.E) = \iiint_{\mathcal{V}} \frac{d}{dt}[\rho(e + \frac{V^2}{2})] \cdot d\mathcal{V} + \iint_S \rho(e + \frac{V^2}{2}) \cdot d\mathbf{S} \tag{2.9}$$

Sammanställ nu 2.7 2.8 och 2.9 för att härleda uttrycket för energiprincipen 2.6:

$$\iiint_{\mathcal{V}} \dot{q}\rho \cdot d\mathcal{V} - \iint_S \rho \mathbf{V} \cdot d\mathbf{S} + \iiint_{\mathcal{V}} \rho(\mathbf{f} \cdot \mathbf{V}) \cdot d\mathcal{V} =$$
$$\iiint_{\mathcal{V}} \frac{d}{dt}[\rho(e + \frac{V^2}{2})] \cdot d\mathcal{V} + \iint_S \rho(e + \frac{V^2}{2}) \cdot d\mathbf{S} \tag{2.10}$$

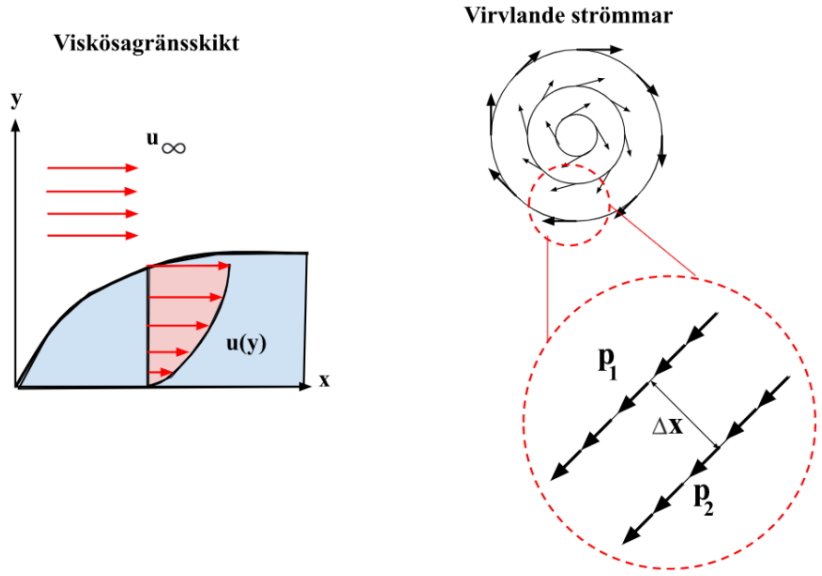
2.2 Viskositet

Viskositet är ett mått på en vätskas förmåga att motstå deformation vid pålagda skjuvkrafter, [8, s. 25]. Till skillnad från solider behåller fluider inte sin form eller struktur vid pålagda skjuvlaster utan deformeras istället kontinuerligt, tills dess att jämvikt uppnås. Som nämnt i bakgrunden 1.1 innefattar denna definition av en fluid både vätskor och gaser och det som skiljer dem åt (bland annat) är deras viskositet. Viskositet är en egenskap som medför till exempel friktion och bildandet av viskösa gränsskikt som Navier-Stokes ekvationerna tar med i konserveringsekvationerna.

Ett mycket fundamentalt koncept är att fluidelement i flöde som är i kontakt med väggar alltid har samma hastighet som väggen det vill säga lika med 0 relativt flödet. En konsekvens av detta är att det bildas en hastighetsgradient på grund av att alla fluider är viskösa. Ju närmare fluidelementen är till ytor desto lägre hastighet har de vilket bildar detta viskösa gränsskikt som dessutom växer vertikalt längs flödets riktning vilket visas i figur 2.2 (vänster). I figuren visas hur gränsskiktet beter sig och att hastigheten "u" varierar med y medans hastigheten utanför gränsskiktet är den så kallade friströmshastigheten "u_i". Utanför viskösa gränsskiktet kan friktionsfri strömning antas där Eulers ekvationer blir relevanta [6] och samtidigt kan experimentella modeller på olika fluiders gränsskikt användas för att räkna på strömningens inverkan på en kropp eller yta. Dessa gränsskikt ger upphov till friktion och luftmotstånd, men de kan också avvika från ytor och leda till virvlande strömning. Att flödet börjar virvla (på Engelska "*vorticity*") och fluktuera leder också till att höga tryckgradienter kan bildas över en mycket kort sträcka, som i figur 2.2 (höger) och detta kommer orsaka problem för simuleringen

vilket kommer att beskrivas senare. Gränsskikten gör det med andra ord allmänt mycket svårare att räkna analytiskt på flöden över mer komplicerade geometrier.

Fenomenet viskositet medför alltså en del svårigheter som endast kan bortses från i ett fåtal idealfall, vilket ökar komplexiteten. Skjuvspänningen, som är en ytterliggare kraft, beror på hastighetsgradienter i fluiden [8, s. 3]. På grund av slumpmässigheten som viskositet introducerar kan inte matematiken hantera dess fluktuationer och det finns inga kända lösningar till rörelsemängds-ekvationen [8, s. 360]. Ekvaktionen måste istället lösas numeriskt.

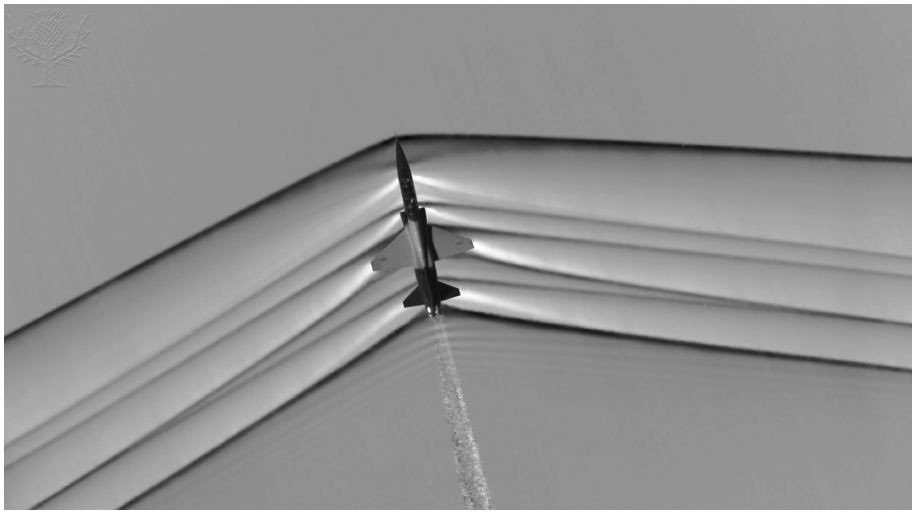


Figur 2.2: Ett växande visköst gränsskikt och virvlande strömmar

2.3 Kompressibelt flöde

Då flödets hastighet är jämförbar med ljudetshastighet medför det att "densitetsskillnaderna i fluiden blir så pass betydande att flödet kallas kompressibelt [8, s. 609, egen översättning]. Värt att notera är för det första att *ljudetshastighet är hastigheten som information kan färdas i ett material*, för det andra att *i ett flöde är ljudetshastighet inte nödvändigtvis konstant* och till sist att *ljudetshastighet beror på bland annat materialet, trycket, temperaturen och flödets hastighet*. Med det sagt definieras Machtalet som proportionaliteten mellan flödets hastighet och ljudetshastigheten i flöde. Ett vanligt och vedertaget antagande är att vid ett Machtal över 0.3 kan flödet antas vara kompressibelt [5, s. 2]. Intressant nog blir det viskösa gränsskiktet mycket tunt vid höga hastigheter och flödet kan i många fall antas vara friktionsfritt (inviscid) [6] och därför används framförallt Eulerekvationerna istället för Navier-Stokes ekvationerna. Vid Mach-tal under ett (M<1) kallas flödet subsoniskt, vid Mach-tal lika med 1 (M=1) är flödets hastighet lika med ljudetshastighet och vid Mach-tal över ett (M > 1) kallas flödet *supersoniskt*.

Ofta används definitionen *transsoniska flöden* för att beskriva flöden där en blandning av supersoniska och subsoniska hastigheter förekommer, runt $0.8 \leq M \leq 1.2$.



Figur 2.3: Stötvågor kring ett Jetflygplan [13]

2.3.1 Supersoniskt flöde

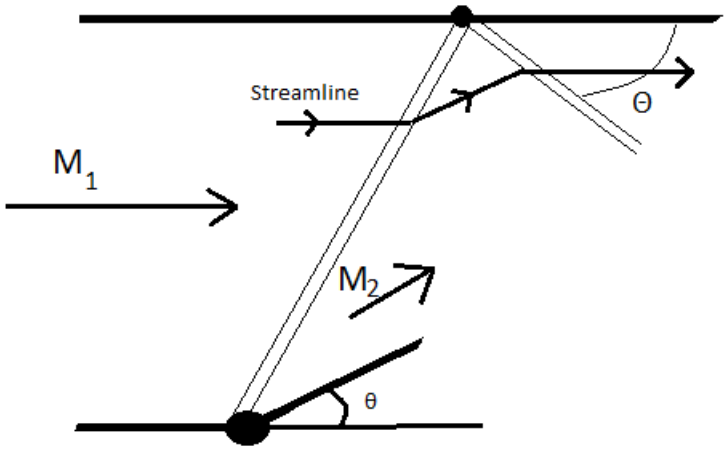
Vid supersoniska flöden rör sig partiklarna snabbare än vad ljudvågor kan överföra information. Det blir alltså omöjligt för en partikel längre uppströms i flödet att få information om vad som sker nedströms. I subsoniska flöden kommer till exempel geometriändringar ge upphov till ljudvågor som propagerar uppströms som kommande partiklar påverkas av. Det är dock inte fallet för ett supersoniskt flöde. Vid hastigheter över Mach 1 rör sig partiklarna i flödet snabbare än vad information kan färdas och för att "lösa" det ger naturen upphov till stötar, se figur 2.3.

2.3.2 Stötar

Rent fysiskt kan stötar föreställas som ett mycket tunt "filter", vid *storleksordningen* 10^{-5}*cm för luft under vanliga förutsättningar* [5, s. 86, egen översättning]. Filtret är bestående av ett lager av partiklar som samlas uppströms efter störningar, till exempel geometriändringar. "Filtret" är i själva verket så tunt att det behandlas som en diskontinuitet där flödets egenskaper, som tryck, hastighet, densitet, temperatur, etc, dramatiskt ändras och entropi ökar. Efter stöten är flödet alltid subsoniskt i stötens normalriktning, enligt termodynamikens andra lag. När flödet är subsoniskt kan det kompensera för störningar, som geometriändringar, då information kan färdas uppströms. För enkla geometrier och flödesfall är teorin för stötar mycket väl etablerad och kompletteras av experimentell data men för mer komplicerade fall är det mycket svårt att lösa analytiskt enligt White: *För att analysera kompressibla flöden krävs flera komplicerade algebraiska ekvationer och de flesta av dessa är mycket svåra att manipulera och invertera.* [8, s. 609, egen översättning].

Det blir alltså mycket mer relevant i allmänhet att hellre lösa dessa fall numeriskt och enligt White: *i dagsläget, med datorernas ökande beräkningskraft och minne, kan nästan alla laminära flöden modelleras med noggrannhet*, [5, s.5, egen översättning].

Ytterligare en egenskap hos stötar är att de kan reflekteras och studsa mot exempelvis väggar [8, s.152]. I figur 2.4 rör sig ett flöde med hastighet M_1. När flödet påträffar en ramp måste flödet ändra riktning, och då flödet är supersoniskt bildas då en stöt. När flödet sedan når taket måste bildas ytterligare en stöt som måste vara tillräckligt stark för att flödet ska bli parallellt med taket, alternativt att stöten reflekteras.



Figur 2.4: Reflektion av en stöt

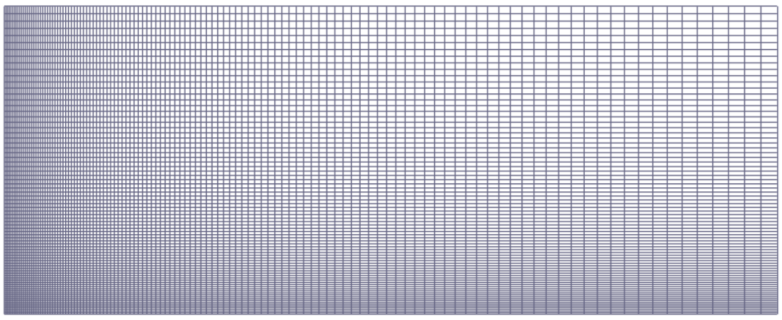
Fenomenet uppenbaras i flera av testfallen.

2.4 Numerisk lösning av Eulerekvationerna

För att få fram flödets egenskaper måste Eulers ekvationer lösas för var och en av punkterna i flödet. Detta är förstås orimligt för en människa att göra analytiskt, och lämpar sig bättre att göras med datorer numeriskt, det vill säga *Computational fluid dynamics* (CFD). Men först måste dessa punkter skapas, och en rutin upprättas och i detta delkapitel förklaras därför hur de flesta CFD program fungerar, inklusive "G3D::flow"

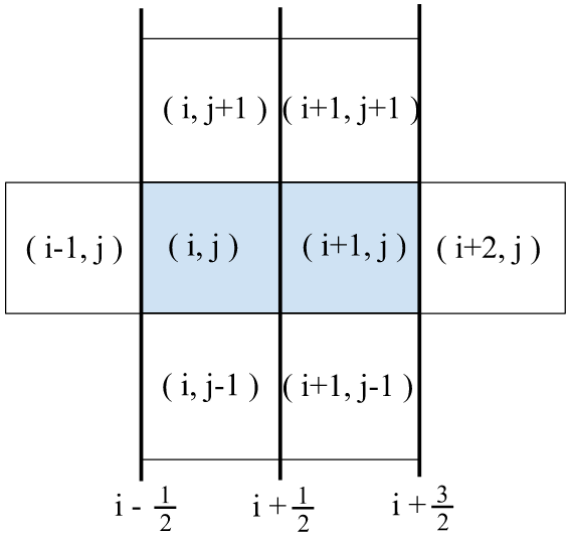
2.4.1 Diskretisering

Första steget är att skapa en diskretisering av volymen som flödet strömmar genom. Nästa steg är att lösa Eulerekvationerna i var och en av cellerna genom att ta medelvärdet av tryck, densitet, hastighet, etc i varje cell. Figur 2.5 visar hur detta kan se ut, notera att cellerna inte behöver vara av samma storlek.



Figur 2.5: Diskretisering i celler

För att utföra beräkningarna används fluxmolekyler, se figur 2.6, som i detta projekt består av en rad med 4 celler i varje riktning där det i mitten av varje rad är en cellvägg. Det är över denna vägg som till exempel ändringen i densitet eller tryck beräknas när stötar undersöks. Cellerna innehåller värden på olika flödesstorheter som används i beräkningar och det finns pekare till dessa i koden. Även cellväggarna har pekare till dess värden som visar saker som randvilkor. Rent fysiskt kan det ses att det som flödar ut från cell (i, j) i x-led måste flöda in i cell (i+1, j), och som det visades i Euler ekvationerna är det massa, rörelsemängd och energi som flödar från en närliggande cell till en annan. Analogt med härledningen 2.1 kan det nu ses som att var och en av cellerna är en mycket liten del av kontrollvolymen och att integrationen över hela domänen egentligen är summeringen av alla celler. Detta implicerar att ju mindre dessa celler blir, desto bättre är antagandet att flödesstorheterna är konstanta inom cellen och desto bättre blir den numeriska approximationen.



Figur 2.6: Fluxmolekyl för användning vid numerisk beräkning

För cell (i, j) kan flödesstorheter skrivas om på lokal form så att egenskaper som flödar in i cellen flödar genom $i - \frac{1}{2}$ och ut genom $i + \frac{1}{2}$. Kontinuitetsekvationen (2.1) kan nu skrivas om så att volymen av cellen ges av $V_i = \iiint_{\mathcal{V}} d\mathcal{V}$, och cellens

area ges av $A_{i+\frac{1}{2}} = \oiint_S d\mathcal{S}$. Observera att variabler med streck över innebär cell-medelvärden över tid.

$$\frac{\partial}{\partial t} \oiiint_{\mathcal{V}} \rho d\mathcal{V} = V_i \frac{d}{dt}(\overline{\rho_i}) \tag{2.11}$$

$$\oiint_S \rho V \cdot d\mathcal{S} = \overline{(\rho u)}_{i+\frac{1}{2}} A_{i+\frac{1}{2}} - \overline{(\rho u)}_{(i-\frac{1}{2})} A_{i-\frac{1}{2}} \tag{2.12}$$

Massbevarings ekvationen skrivs nu som:

$$V_i \frac{d}{dt}(\overline{\rho_i}) - \overline{(\rho u)}_{(i-\frac{1}{2})} A_{(i-\frac{1}{2})} + \overline{(\rho u)}_{i+\frac{1}{2}} A_{i+\frac{1}{2}} = 0 \tag{2.13}$$

På liknande vis skrivs de resterande Euler ekvationerna om på FVM form (rörelsemängd respektive energi konservering):

$$V_i \frac{d}{dt}\overline{(\rho u)}_i = \overline{(\rho u^2 + p)}_{i-\frac{1}{2}} A_{i-\frac{1}{2}} - \overline{(\rho u^2 + p)}_{i+\frac{1}{2}} A_{i+\frac{1}{2}} + \overline{p_i}(A_{i+\frac{1}{2}} - A_{i-\frac{1}{2}}) \tag{2.14}$$

$$V_i \frac{d}{dt}\overline{(\rho e_o)}_i = \overline{(\rho u h_o)}_{i-\frac{1}{2}} A_{i-\frac{1}{2}} - \overline{(\rho u h_o)}_{i+\frac{1}{2}} A_{i+\frac{1}{2}} \tag{2.15}$$

Notera att alla termer med suffixen "i" är termer som tillhör cell "i" medans termer med suffixen "$i + \frac{1}{2}$" eller "$i - \frac{1}{2}$" tillhör cell väggen. Notera även konserveringen av flödesstorheter mellan två närliggande celler så att, till exempel, massan som flödar ut ur en cell flödar in i en närliggande cell:

Cell (i): $V_i \frac{d}{dt}\overline{(\rho e_o)}_i = \overline{(\rho u h_o)}_{i-\frac{1}{2}} A_{i-\frac{1}{2}} - \overline{(\rho u h_o)}_{i+\frac{1}{2}} A_{i+\frac{1}{2}}$
Cell (i+1): $V_{i+1} \frac{d}{dt}\overline{(\rho e_o)}_{i+1} = \overline{(\rho u h_o)}_{i+\frac{1}{2}} A_{i+\frac{1}{2}} - \overline{(\rho u h_o)}_{i+\frac{3}{2}} A_{i+\frac{3}{2}}$

Denna konserveringsprincip är ytterst viktig för korrekt hantering av stötar angående stötens position i flödet, styrka och hastighet, och är därmed ytterst relevant för detta projekt. För numeriska beräkningen väljs nu kvantiteterna ($\overline{\rho_i}$, $\overline{\rho u}$, $\overline{\rho e_o}$) som de primära okända och genom att möblera om ekvation (2.13) - (2.15) löses nu följande system av ordinära differentialekvationer.

$$\frac{d}{dt}(\overline{\rho_i}) = \frac{1}{V_i}[\overline{(\rho u)}_{(i-\frac{1}{2})} A_{(i-\frac{1}{2})} - \overline{(\rho u)}_{i+\frac{1}{2}} A_{i+\frac{1}{2}}] \tag{2.16}$$

$$\frac{d}{dt}\overline{(\rho u)}_i = \frac{1}{V_i}[\overline{(\rho u^2 + p)}_{i-\frac{1}{2}} A_{i-\frac{1}{2}} - \overline{(\rho u^2 + p)}_{i+\frac{1}{2}} A_{i+\frac{1}{2}} + \overline{p_i}(A_{i+\frac{1}{2}} - A_{i-\frac{1}{2}})] \tag{2.17}$$

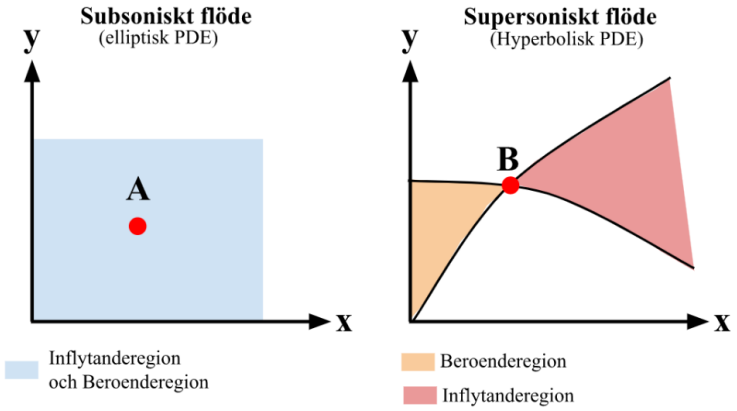
$$\frac{d}{dt}\overline{(\rho e_o)}_i = \frac{1}{V_i}[\overline{(\rho u h_o)}_{i-\frac{1}{2}} A_{i-\frac{1}{2}} - \overline{(\rho u h_o)}_{i+\frac{1}{2}} A_{i+\frac{1}{2}}] \tag{2.18}$$

2.4.2 Time-Marching

Med diskretiseringen som presenteras i kapitel 2.4.1 kan de ordinära differentialekvationerna lösas numeriskt för att få ut alla flödesstorheter. Det finns dock ett antal kvarstående svårigheter med simuleringen och det är framförallt att karaktären hos de ordinära differentialekvationerna är olika beroende på om flödet är subsoniskt eller supersoniskt.

Problemet härstammar från hur information i ett flöde kan sprida sig och därmed hur punkter i flödet påverkar och påverkas av andra punkter i flödet. Information sprider sig i ljudetshastighet och för ett stationärt (icke-tidsberoende) subsoniskt flöde innebär detta att en punkt A i flödet påverkas av alla andra punkter i flödet eftersom information kan sprida sig uppströms. Det innebär dessutom att punkten A påverkar alla andra punkter i flödet, se figur 2.7. För ett stationärt supersoniskt flöde gäller att en punkt B i flödet endast påverkas av punkterna uppströms i flödet, en region som kallas beroenderegionen (eng. region of dependence). Det gäller dessutom att punkt B endast kan påverka punkter nedströms i flödet, en så kallad inflytanderegion (region of influence). De styrande differentialekvationerna för ett subsoniskt flöde kallas elliptiska och för supersoniska flöden kallas de ekvationerna hyperboliska. För hyperboliska ekvationer finns en unik lösning för en punkt i flödet givet gränsvillkoren. Lösningen till en punkt för elliptiska ekvationer beror dock på lösningen av alla andra punkter i flödet. Beroende på om flödet är supersoniskt eller subsoniskt måste simuleringen alltså behandla lösningen av differentialekvationerna olika för ett stationärt flöde, enligt Anderson:

Att beteendet av de styrande ekvationerna ändras från elliptiska till hyperboliska över ljudbarrieren orsakar allvarliga matematiska och numeriska svårigheter - faktum är att att stationära lösningar av subsoniska och supersoniska regioner i flödet vanligtvis behandlas separat och sedan på något sätt sammanfogas i det transoniska området nära ljudlinjen. [5, s. 435, egen översättning]



Figur 2.7: Informationsregioner för supersoniska och subsoniska flöden

Det finns dock en annan lösning till detta problem som inte innebär att behandla det subsoniska- och supersoniska områdena separat och sedan "sy ihop" dem där de möts som White nämnde. En mer elegant lösning är att börja simuleringen med antagandet att flödet är instationärt (varierar med tid) givet godtyckliga initial villkor, även om simuleringen egentligen ska hantera ett stationärt flöde. Det gäller att instabila flöden styrs av *hyperboliska* ekvationer för *både* subsoniska och supersoniska flöden och därför kan differentialekvationerna lösas på samma sätt i alla punkter [5, s. 389]. Tekniken som används är den så kallade "Time marching technique" där flödet löses iterativt i tidssteg tills förändringarna i flödetsstorheter asymptotiskt minskar till noll och simuleringen konvergerar till en stabil lösning. G3D:flow.py använder den så kallade Runge-Kutta metoden för att avancera lösningen i tid och är en enkel algoritm med en bra balans mellan noggrannhet och beräkningskostnad. När flödet är fullt utvecklat kan data nu insamlas för att få ut flödesstorheter i alla punkter i flödet.

2.4.3 Runge-Kutta

Med Runge-Kutta metoden kan alla Euler ekvationer lösas med avseende på tid för att simulera flödets egenskaper vid varje punkt i flödet (varje cell) och på så vis få en bra uppfattning om hur flödet kommer att bete sig över godtyckliga geometrier för både subsoniskt och supersoniskt flöde.

För att förenkla notationen skrivs ekvation (2.16) - (2.18) om på vektor form som i (2.19):

$$\frac{d}{dt}\mathbf{Q} = \mathbf{F}(\mathbf{Q}) \tag{2.19}$$

För att simulera flödet med avseende på tid skall beräkningarna utföras i tidssteg och därmed används en enkel algoritm för att gå från $\mathbf{Q}(t_n)$ till $\mathbf{Q}(t_{n+1})$ med tidssteget Δt:

$$\mathbf{Q}^* = \mathbf{Q}(t_n) + \Delta t\mathbf{F}(\mathbf{Q}(t_n)) \tag{2.20}$$

$$\mathbf{Q}^{**} = \mathbf{Q}(t_n) + \frac{1}{2}\Delta t\mathbf{F}(\mathbf{Q}(t_n)) + \frac{1}{2}\Delta t\mathbf{F}(\mathbf{Q}^*(t_n)) \tag{2.21}$$

$$\mathbf{Q}(t_{n+1}) = \mathbf{Q}(t_n) + \frac{1}{2}\Delta t\mathbf{F}(\mathbf{Q}(t_n)) + \frac{1}{2}\Delta t\mathbf{F}(\mathbf{Q}^{**}(t_n)) \tag{2.22}$$

Denna Runge-Kutta lösare är av tredje ordningen i och med att den utför tre steg för att komma fram till en slutlig lösning. Först beräknas en mellanliggande lösning med hjälp av initialvärden och de styrande Euler-ekvationerna för att uppskatta lösningens derivata. I andra steget används denna lutningsuppskattningen från första steget för att beräkna en mellanliggande lösning i mitten av tidssteget Δt. Slutligen används lutningsuppskattningen från andra steget till att lösa ut $\mathbf{Q}(t_{n+1})$ och sedan kan dessa tre steg repeteras för att beräkna $\mathbf{Q}(t_{n+2})$, $\mathbf{Q}(t_{n+3})$, etc. Som konsekvens av denna metodik händer det dock att vissa lösningar inte konvergerar och det sägs att simuleringen då blir instabil, något som kan

orsakas när simuleringen försöker hantera stötar. Detta problem åtgärdas av en teknik som ofta kallas "shock capturing" men först är det viktigt att förstå hur "time-marching" tekniken kan konfigureras med Δt.

2.4.4 Tidsstegs-konfigurering

Utifrån Runge-Kutta algorithmen (2.20 - 2.22) kan det ses att ju mindre tidssteg som används, Δt, desto fler tidsstegs beräkningar behövs för att simulera över totala tiden t. I och med att lösaren använder gradienter för att beräkna nästa steg i Runge-Kutta algoritmen kan det också förstås att mindre tidssteg leder till mindre fel mellan varje steg i lösaren och därmed blir lösningen mer noggrann. Förhoppningsvis ger detta en rätt intuitiv förståelse av varför mer noggrannare simuleringar tar längre tid att köra, det vill säga ökar beräkningskostnaden. Men precis som varje cell inte behöver vara lika stor i meshen (som i figur 2.5), behöver tidssteget i varje cell inte vara konstant och detta leder till en mycket snabbare konvergering av lösningen. I och med att testfallen som kommer genomföras har olika tidsstegskonfigurering är det relevant att förstå när man kan variera tidsteget i varje cell och när detta inte kan göras då det leder till divergens, mer om detta i kapitel 2.6.

För detta projekt är tre olika konfigureringar relevanta: "local time-stepping", "time-accurate (constant)" och "time-accurate (CFL)"."Local time-stepping" är när varje cell kan ha olika tidssteg och detta görs när flödet är stationärt (icke-tidsberoende). I och med att alla celler kommer konvergera till samma lösning spelar det inte roll om en cell har kommit fram till den lösningen snabbare än en annan och det blir därför användbart att ha stora tidssteg för celler som inte ändras mycket (små gradienter) för varje tiddsteg, och mindre tiddsteg där mycket ändring sker (stora gradienter). För icke-stationära flöden (tidsberoende) måste "time-accurate" lösningar användas där alla celler har samma tidssteg för att mer noggrant simulera de tidsberoende egenskaperna av flödet. Antingen kan man sätta tidsteget till ett konstant värde som är tillräckligt litet för varje cell ("time accurate (constant)") eller kan man använda Courant-Friedrichs-Lewy (CFL) kriteriet som ser till att tidssteget alltid är litet nog för att undvika divergens. Detta kriteriet ser ut som i ekvation 2.23:

$$CFL = \frac{(u + c) * \Delta t}{dx} \leq 1 \tag{2.23}$$

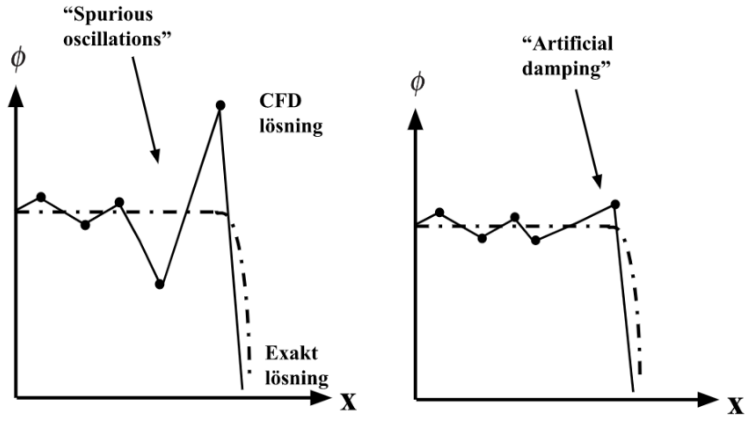
Här är u flödets hastighet i x-led, c ljudhastigheten, dx cellens bred och Δt tidssteget. Om ekvationen möbleras om som i ekvation 2.24 ges nu kravet på hur stort Δt får vara och det ses att tidssteget måste vara så pass litet att en partikel med hastigheten u på tiden Δt inte kan färdas längre än sträckan dx. Ett fysiskt sätt att förstå detta är att en partikel aldrig får hoppa över en cell och detta åstadkoms genom att låta Δt uppfylla CFL kriteriet.

$$\Delta t \leq \frac{dx}{u + c} \tag{2.24}$$

Notera att CFL kriteriet också används för lokalt varierande tiddstegsmetoder men att varje cell kan ha olika tidsteg så länge de uppfyller CFL kriteriet. För "time-accurate" lösare måste alla celler ha samma tidssteg och därför begränsas lösningen mycket av att celler är mindre och enligt CFL kriteriet måste ha mycket mindre tidssteg.

2.5 *Shock capturing*

Som nämnt i beskrivningen av Runge-Kutta algoritmen 2.4.3 löser simuleringen differentialekvationerna genom att räkna med gradienter i varje steg men detta blir förstås problematiskt när gradienterna är nästan oändliga, som för stötar och därför måste man "fånga" stötarna. *Shock capturing* är en av flera metoder för att identifiera och behandla stötar i numeriska lösningar av flödes-problem. Till skillnad från *shock-fitting* krävs ingen tidigare kunskap om varken var eller hur många stötar som kommer att framträda för att det ska fungera, hävdar J.D. Anderson Jr. [5, s. 422]. Enligt samma författare [5, s. 422] kommer stötvågen att visa sig som en region av stora gradienter, smetad över ett antal celler. Som förklarat i delkapitel 2.3 kan stötar antas vara diskontinuerliga, vilket innebär att flödesstorheter som till exempel tryck, hastighet och temperatur drastiskt ändrar värden över en mycket kort sträcka vilket ger upphov till mycket, nästan oändligt, stora gradienter. Det är just dessa gradienter som kan orsaka problem för simuleringen och ge upphov till *spurious oscillations*[6], se figur 2.8, som visar hur numeriska simuleringar med hjälp av högre ordningens lösare, i detta fall Runge-Kutta, över- eller underskattar faktiska värden som kan leda till att lösningen inte konvergerar.



Figur 2.8: Numeriska fel vid skarpa gradienter

För att åtgärda detta implementerar G3D:flow.py artificiell dämpning som i figur 2.8 för att lösaren mer exakt ska kunna hantera stötar. I figuren används ϕ för att representera flödesstorheter som tryck, eller temperatur och här syns det att en stöt ger upphov till en drastisk ändring av denna storhet över en mycket kort sträcka i x-led som simuleringen har svårt att följa. Detaljerna för hur exakt

detta går till är inte speciellt relevanta men i detta fall implementeras lokala energiförluster (dissipative terms) som dämpar ner svängningarna [1, s. 3]. För att göra detta måste simuleringen först kunna identifiera stötar och här finns flera varianter av sensorer men mest relevant till detta projekt är *Jameson-sensorn* och *Ducros-sensorn*.

2.5.1 *Jameson-sensorn*

Jameson-sensorn består i en enkel storhet baserad på trycket i en specifik cell samt trycket i omgivande celler och är designad för att upptäcka diskontinuiteter i tryck, och sedan anpassa det till omgivningen [1, s. 1]. Sensorn Ψ beräknas genom följande, se kod (appendix A.2)

$$\nu_{i,j} = \frac{|p_{i+1,j} - 2p_{i,j} + p_{i-1,j}|}{p_{i+1,j} + 2p_{i,j} + p_{i-1,j}} \tag{2.25}$$

$$\texttt{Jameson} = \Psi_{i+1/2,j} = max(\nu_{i,j}, \nu_{i+1,j}) \tag{2.26}$$

Observera att för $\nu_{i,j}$ används trycket i cell (i, j), (i-1, j) och (i+1, j), det vill säga, simuleringen undersöker tryckskillnader i de närliggande cellerna. Sedan beräknas $\nu_{i+1,j}$ genom att ta trycket i cellerna (i+1,j), (i, j) och (i+2, j). Med andra ord jämför simuleringen trycket för hela raden "j" i fluxmolekylen, se figur 2.6. Sedan tas den största ν för att få ut $\Psi_{i+1/2,j}$, alltså en tryckfaktor på den cellyta som ligger i mitten av fluxmolekylen. Beroende på hur trycket i de olika cellerna varierar kan simuleringen härmed avgöra om en stöt har uppstått mellan celler och om dämpning behöver implementeras. Detta är en så kallade tryckbaserad sensor (pressure based). Om tryckskillnaden är stor nog antar lösaren att en stöt har påträffats och applicerar sedan artificiell dämpning som gör att simuleringen bättre avspeglar verkliga fysikaliska resultat och förebygger ostabilitet i simuleringen.

2.5.2 *Ducros-sensorn*

Enligt H.S.Khalsi som refererar till F.Ducros och E.Garnier *kan diskontinuiteter i tryck uppstå såväl i turbulenta fluktuationer som i stötar*[10, s. 24, egen översättning]. Därför kan det hända att Jameson-sensorn ibland misstar turbulenta fluktuationer för stötar. I figur 2.2 (höger) ges ett exempel på hur tryckskillnader mellan P_1 och P_2 i en virvel kan vara tillräckligt stora för att missidentifieras som en stöt av Jameson-sensorn enligt ekvation 2.25. Då sensorn aktiveras introduceras dämpning där den egentligen inte borde och detta leder till att för mycket energi tas ur systemet och påverkar uppskattningen av turbulens mycket negativt:

"The use of an arbitrary dissipative numerical method (the standard Jameson's artificial viscosity for the present study) can lead to a fairly good picture of the flow (increase of energy, of transverse vorticity fluctuations and continuity of streamwise vorticity fluctuations through the shock). But this picture is not correct far from the shock where the large artificial dissipation affects turbulence more than the subgrid scale model, leading to a wrong prediction of kinetic energy dissipation....The use of the sensor $\Psi\Phi$ to recover

a reliable prediction of kinetic energy decay of turbulence out of the shock leads to better results." [4, s. 545]

För att kunna skilja stötar från turbulenta fluktuationer och virvlande introducerar F.Ducros Et.al. [4, s. 524] den så kallade Ducros-sensorn, ekvation 2.27. Notera att Ducros-sensorn består av Φ (2.28) som multipliceras med Jameson-sensorn 2.26.

$$\Psi_{i+0.5,j}, \Phi_{i+0-5,j} = max(\Psi_{i,j}\Phi_{i,j}, \Psi_{i+1,j}\Phi_{i+1,j}) \tag{2.27}$$

Mer explicit ser uttrycket för Φ ut på följande sätt

$$\Phi = \frac{(\nabla \cdot \boldsymbol{u})^2}{(\nabla \cdot \boldsymbol{u})^2 + \omega^2 + \epsilon} \tag{2.28}$$

Med $\epsilon = 10^{-30}$, lokala hastighetsvektorn u och vorticiteten ω som i ekvation 2.29

$$\omega = \nabla \times \boldsymbol{u} \tag{2.29}$$

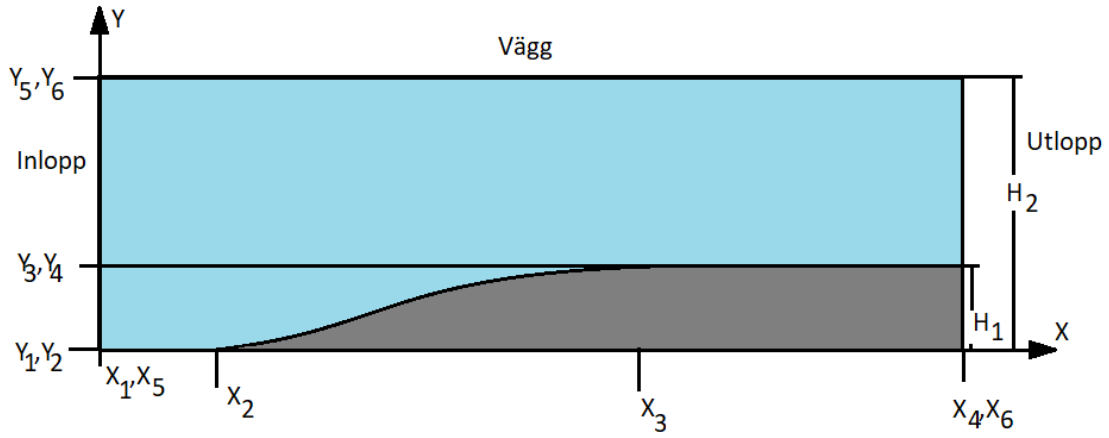
Notera att ϵ endast tilläggs för att undvika att simuleringen delar med noll i någon beräkning. Med den nya termen blir sensorn mer känslig för den lokala vorticiteten och kan beroende på denna anpassa hur mycket dämpning som skall implementeras för att inte orsaka onödiga energiförluster i systemet.

2.6 Testfall och Geometri

I följande delkapitel beskrivs de testfall som utfördes och vilken in-data och konfigurering som användes för att utföra testen så att testen ska kunna återskapas. I detta projekt användes fyra olika testfall, nämligen: *ramp*, *Q1DNozzle*, *Q1DShockTube* och *Shock - Boundary Layer Interaction*, varav den sista angår visköst flöde.

2.6.1 Ramp konfiguration

Första testfallet som ska användas är på en ramp. Geometrin hos detta testfall kan ses i figur 2.9, man har ett supersoniskt in-flöde som träffar en ramp som börjar en bit in i geometrin.



Figur 2.9: Geometri för ramp-testet

Mått som endast betecknas med symboler i figur 2.9 finns i tabell 2.1.

Tabell 2.1: Tabell med geometri till Ramp testfall

Variabel	Värde
X_1	0.0
X_2	1.0
X_3	4.0
X_4	7.0
X_5	0.0
X_6	7.0
Y_1	0.0
Y_2	0.0
Y_3	0.6
Y_4	0.6
Y_5	2.0
Y_6	2.0
H_1	0.6
H_2	2.0

Utöver måtten till geometrin tillhandahålls även begynnelsevärden och randvillkor i tabell 2.2.

Tabell 2.2: Tabell med konfiguration till Ramp testfall

Flödeskonstant	Inflöde	Utflöde
ρ	0.267124	0.4045
ρu	138.266	184.242
ρv	0	0
ρw	0	0
Energi	67734	99059

I detta testfall används dessutom "*local time-stepping*" vilket innebär att tidsstegen varierar i varje cell. Detta är eftersom detta testfall angår ett stationärt flöde.

2.6.2 Ramp, förväntat utfall

Den böjning som krävs för att ändra flödets riktning kommer att orsaka en stöt. Stöten behövs för att böja upp flödet och ändra hastighetsvektorns riktning. När rampen sedan blir platt så behöver flödet böjas tillbaka till en horisontal flödesriktning, vilket orsakar ytterligare en stöt. Detta i enlighet med M. White som påstår att när ett supersoniskt flöde stöter på en gradvis ramp kommer flera stötar att bildas, som längre upp från rampen sammanstrålar till att bilda en icke-isentrop sned stöt [8, s.670]. White påstår vidare att när flödet böjs att åter bli horisontellt igen kommer ett antal expansionsvågor att bildas.

Till skillnad från White's exempel finns det i det fall som simuleringen behandlar även ett tak att ta hänsyn till. Stötarna kommer då att, också enligt White, reflekteras från taket [8, s.693]. Detta kommer troligtvis att bilda som ett sick-sack-mönster genom ovansidan av rampen.

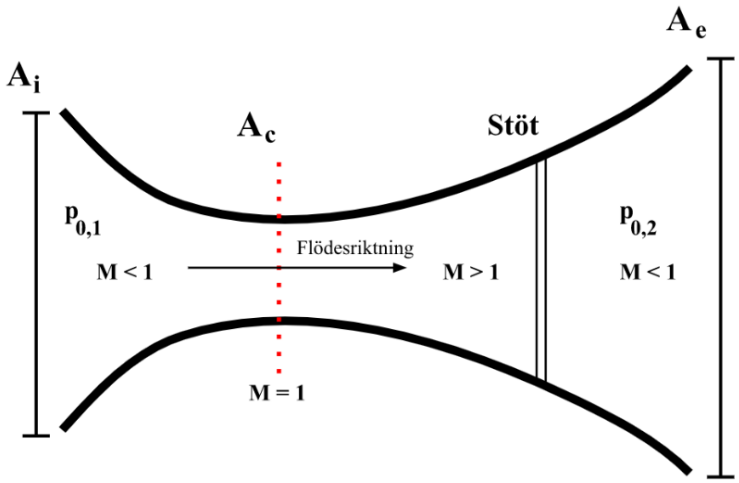
2.6.3 Q1DNozzle, konfiguration

Detta testfall simulerar ett subsoniskt flöde som flödar genom ett konvergerande-divergerande munstycke där även utflödet är subsoniskt som i figur 2.10. Att det kallas "Q1D" står för *kvasi-endimensionellt* (eng. Quasi) och innebär att flödet simulerats som om en tvärsnittsarea varierar, men egentligen simuleras bara en dimension. Munstycket börjar med en radie på 0.3 m som sedan konvergerar till 0.25 m och sedan divergerar till en radie på 0.5 m.

Tabell 2.3: Tabell med konfiguration till Q1DNozzle testfall

Flödeskonstant	Inflöde	Utflöde
p_0	101325	$0.6 \cdot 101325$
T_0	288	

I detta testfall används "*local time-stepping*" igen eftersom det är känt att flödet är stationärt.



Figur 2.10: Visualisering av Kvasi-endimensionerlig munstycke

2.6.4 Q1DNozzle, förväntat utfall

För att få ett flöde genom munstycket måste trycket i inloppet vara högre än trycket i utloppet. Då flödet är subsoniskt leder ett minskande tvärsnitt till att flödet accelereras och om tryckskillnaden mellan inloppet och utloppet är stort nog kan flödet bli supersoniskt vid en kritisk area. För supersoniska flöden leder en expanderande tvärsnittsarea till ytterligare acceleration av flödet [8, s. 623]. Beroende på vad utloppets tryck är kan denna expansion leda till att en stöt bildas någonstans i munstycket (som i detta testfall) och denna stöt gör att flödet i utloppet också blir subsoniskt. I och med att flödet accelereras till platsen där stöten sker gäller det att ett flertal andra flödesegenskaper måste minska, nämligen densitet, temperatur och tryck [5, s. 205]. Efter stöten minskar Machtalet markant och det leder till att densitet, tryck och temperatur ökar. Dessa påståenden är baserade i existerande litteratur som [5] och [8] men just detta testfall är dessutom väletablerat och har en analytisk lösning.

2.6.5 Q1DShockTube, konfiguration

Som testfallet ovan är detta test också en kvasi-endimensionell simulering. Detta testfall går ut på att trycksätta ett rör som är slutet i båda ändarna där röret är indelat i två halvor. Simuleringen låter första halvan ha 5 gånger högre tryck än andra halvan medans alla andra tillstånd är lika för båda delar av röret. När simuleringen börjar bildas en stöt på grund av tryckskillnaden mellan delarna. I verkliga experiment delas röret in i två delar separerade av ett membran där en stor nog tryckskillnad får membranet att brista och en stöt att bildas, men i simuleringen krävs endast att testet börjar med en stor tryckskillnad. Röret har en radie på 0.005 m och en längd på 1 m.

Tabell 2.4: Tabell med konfiguration till Q1DShocktube testfall

Flödeskonstant	Inflöde	Utflöde
p	$5 \cdot 101325$	101325
T	300	300

Detta testfall använder "*time-accurate (constant)*" konfigurationen med ett tidssteg på 0.000001 sekunder. Detta görs eftersom stöten inte står still, utan färdas längs röret och sedan reflekteras fram och tillbaka då den studsar mot rörets slutna ändar. I och med att detta därför är ett instationärt flöde (tidsvarierande), lämpar det sig att använda ett "time-accurate" tidssteg.

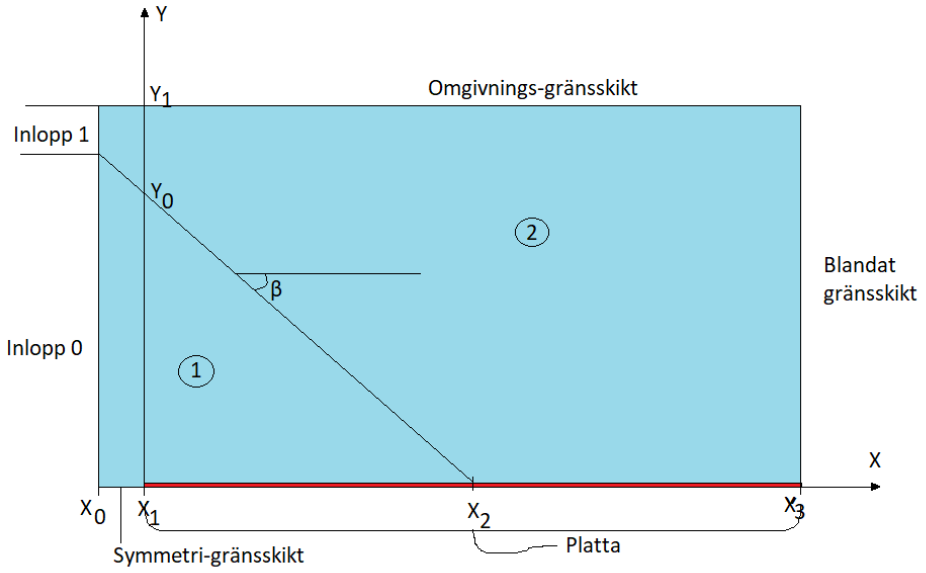
2.6.6 Q1DShockTube, förväntat utfall

Detta test grundar sig i ett mycket vanligt verkligt experiment då stötar ska simuleras, mer specifikt används experimentet för att modellera hur stötar från till exempel explosioner påverkar material [11] eller allmänt hur supersoniska stötvågor beter sig [9]. Stötröret är indelat i två delar där den delen som blir trycksatt kan kallas den "drivande" delen medans delen med atmosförtryck kan kallas den "drivna" delen. När membranet brister färdas stötvågen mot den drivna delen av röret (som ursprungligen hade lägre tryck) medans expansionsvågor ("expansion fans") färdas mot den drivande delen av röret [11]. I två olika CFD experiment för att simulera detta stötvåg visades hur trycket varierar i stötröret (x-led) efter en kort tid, nämligen att det är mycket högt i ena änden (drivande delen) som sedan minskar relativt linjärt tills den när en platå. Efter detta minskar trycket markant och diskontinuerligt i och med att trycket är mindre över stöten. Rent fysiskt kan detta tolkas som att trycket på den drivna sidan av stöten är konstant eftersom informationen av att membranet har brustit inte har nått dit än och att stöten höjer trycket bakom sig. Densiteten plottades också och visade sig ha liknande beteende, alltså att den var högre i den drivande sidan, men minskade i diskontinuerliga "trappsteg", förmodligen på grund av expansionsvågor normalstöten [9] [3].

En annan konsekvens av att testfallet är tidsberoende är att residualen (kvarstående felet i simuleringen) inte kommer att gå ner på samma sätt som för ett stationärt flödesfall (som exempelvis nozzle-fallet ovan). När det gäller detta specifika fall kommer de stöt- och expansionsvågor som skapas då testet börjas att färdas fram och tillbaka i röret och interagera med varandra tills dess att jämvikt uppnås och då blir det ett stationärt tillstånd utan flöde. Eftersom det intressanta är lösarens förmåga att beskriva de stöt- och expansionsvågor som uppkommer tillåts lösaren bara räkna ett fåtal iterationer (efter redan några tusen iterationer kommer vågorna att ha interagerat och det blir omöjligt att avgöra vilken våg som är vilken). Det kommer i detta testfall med andra ord se ut som om simuleringen har fastnat, men detta fall är konfigurerat att stanna efter ett mindre antal iterationer.

2.6.7 Laminärt visköst flöde över en platta, konfiguration

I detta testfall strömmar ett supersoniskt laminärt flöde över en platta och över denna platta bildas, på grund av viskositet, ett mycket tunt gränsskikt. Vid ett annat inlopp över plattan strömmar ett flöde med en annan riktning och hastighet vilket orsakar en stöt som reflekteras i gränsskiktet. I simuleringen bildas denna stöt helt enkelt genom att sätta olika tryck och hastighet på den del av inloppet som representerar flödet innan stöten och den del av inloppet som representerar flödet efter stöten (område 1 och två i figur 2.11) och ända sättet att åstadkomma detta med följande figurering är om det bildas en stöt vid denna punkt. I följande tabeller kan det ses indatan som användes för att genomföra testet.



Figur 2.11: Geometri för testfall *laminärt visköst flöde över en platta*

Tabell 2.5: Beskrivning av geometri

Geometri	Värde
β	30.8°
Inlopp [m]	
$y_0 = 1.2\tan(\beta)$	0.7153
y_1	1
Platta [m]	
x_0	−0.2
x_1	0
x_2	1
x_3	2

Tabell 2.6: Värden för flödeskonstanter

Flödeskonstant	Inlopp 0	Inlopp 1	Utlopp
M	2.15	2.01	2.01
T	$288.00K$	$306.87K$	$306.87K$
p	$0.01bar$	$1.26bar$	$1.26bar$
ρ	$1.23kg/m^3$	$1.44kg/m^3$	$1.44kg/m^3$
a	$340.17m/s$	$351.14m/s$	$351.14m/s$

Detta testfall använder "*time-accurate (CFL)*" som tidsstegskonfiguration där alla celler har samma tidssteg och detta tidssteg beräknas med hjälp av CFL kriteriet i ekvation 2.23. Detta är på grund av att det viskösa gränsskiktet interagerar med stöten och gör simuleringen tidsberoende och därmed kräver en "*time-accurate (CFL)*" konfigurering.

2.6.8 Laminärt visköst flöde över en platta, förväntat utfall

Detta testfall grundar sig i ett experiment utfört av Degrez and C. H. Boccadoro and J.F Wendt med syftet att utforska hur stötar interagerar med viskösa gränsskikt [2]. I och med att detta numeriska testfall är baserat på ett etablerat experiment, se Schlieren visualiseringen i [2, s. 252, fig 4], finns ett antal observationer som borde märkas tydligt i simuleringen. Först är att en stöt bildas med en vinkel β relativt den horisontella plattan i figur 2.11. Som förklarat i 2.3.2 kan stötar reflekteras över ytor och detta är precis det som hände i Degrez-experimentet. Mer relevant för detta arbete är dock att studera hur stöten interagerade med det viskösagränsskiktet. Som förklaras i kapitel 2.2 bildas viskösa grässkikt i flöden som strömmar över ytor, och dessa gränsskikt växer dessutom kontinuerligt över ytan. I experimentet som utfördes "kolliderar" stöten med gränsskiktet vilket får den att minska i storlek, och enligt studien bildas också virvlande strömmar i gränsskiktet. Angående detta projekt är det förväntat att den nya sensorn undviker att felaktigt identifiera virvlar som stötar och implementerar mindre artificiell dämpning än originalsensorn.

3
Metodik

I den här sektionen beskrivs det allmänna arbetsflödet för projektet. Här gjordes ett beslut att dela in projektets arbete i två faser nämligen validering och utveckling. I och med att den nya Ducros-sensorn skulle implementeras måste dess funktionalitet först valideras mot Jameson-sensorn i testfall där det är känt att Jameson-sensorn fungerar, alltså för de icke-viskösa testfallen. Efter detta var gjort och det konfirmerats att den nya sensorn beter sig som den ska testades den på det viskösa testfallet där helt nya resultat förväntades. Att det ens finns en fas två beror på att Ducros-sensorn funkade som den skulle för de icke-viskösa testfallen, mer om detta i resultatdelen (Kapitel4). Det är känt att Jameson-sensorn inte riktigt funkar som den ska för viskösa flöden och resultaten från den nya sensorn behövdes därför analyseras mot litteratur och experimentell data. Utöver detta kunde nya sätt att implementera Ducros-sensorn utforskas i syftet att förbättra den (minska beräkningskostnad). Med detta sagt kändes det naturligt att dela upp metodiken i dessa två faser av projektets arbete, den första delen som bara angår de icke-viskösa testfallen och den andra som berör det viskösa testfallet. Med detta sagt utfördes följande arbetsgång:

- Icke-viskösa flöden: Validering

    1. Inventering av teori och nuvarande behandling av stötar
    2. Inventering av ny teori och ny behandling av stötar
    3. Implementering av ny sensor i kod
    4. Utförandet av icke-viskösa testfall
    5. Analys av resultat och jämförelse med gamla metoder i koden, extern experimentell data och känd teori

- Viskösa flöden: Insamling av ny data

    1. Utforskning av alternativa beräkningsmetoder för den nya sensorn
    2. Implementering av alternativa metoder i koden
    3. Utförandet av viskösa testfall med nya konfigureringar
    4. Analys och jämförelse av gamla och nya sensorn för det viskösa fallet och jämförelse med experimentell data och känd teori
    5. Jämförelse av beräkningskostnad av de alternativa konfigureringarna för den nya sensorn jämfört med den gamla sensorn

3.1 Stöthantering i icke-viskösa flöden

Som nämnt i kapitel 2 framkom ett antal problem med simuleringen då den försöker lösa stötar och därför implementeras stöthantering. Det beskrevs att det fanns en nuvarande lösning, den så kallade "Jameson-sensorn" som används för att identifiera stötar men att denna sensorens prestanda är bristande för viskösa flöden. Det skall därför implementeras en ny sensor och i detta kapitel förklaras det hur denna sensorn valdes, implementerades och testades. För att testa att den nya sensorn fungerar minst lika bra som den gamla jämfördes de för icke-viskösa testfall där de borde producera samma simuleringsresultat.

3.1.1 Inventering av föregående stöthantering

För att inleda denna del av utvecklingen undersöktes hur den dåvarande lösningen fungerade. Första steget i denna undersökning var att identifiera var i koden stöthanteringen sker. Genom inventeringen erhölls en större förståelse för hur programvaran är uppbyggd och hur de olika delarna fungerade.

Det noterades att om variabeln "fnctp" sattes till $fnctp > 0$ innebar det att stöthantering sker i filen "flux.py" varav nästa steg blev att förstå de ingående variablerna och dess betydelse. Det konstaterades att beräkningen av Jameson-sensorn görs på varje "fluxmolekyl" och att den gör detta med hjälp av att jämföra trycket i varje närliggande cell. Idéen med detta är att om tryckskillnaden är tillräckligt stor kan det antas att en stöt att uppkommit. Sensorn beräknas enligt 2.5.1, ν beräknas på båda sidor om cellytan i centrum av fluxmolekylen och av dessa fås sedan Ψ som är det största värdet på ν. Denna sensor, mer specifikt värdet av den, användes sedan som en konstant i de olika jämviktsekvationerna för massa, energi, impuls etc för att få rätt dämpning på stöten. Koden kan ses i Appendix A.1.

3.1.2 Inventering av ny stöthantering

I samråd med handledare kom gruppen fram till att en "Ducros-sensor" skulle implementeras. Ducros är en ytterligare sensor som beräknas på var och en av flux-molekylerna och som bygger vidare på Jameson-sensorn. Ducros-sensorn tar hänsyn till viskositeten i flödet och beräknas med hjälp av divergensen av flödeshastigheten samt vorticiteten, se 2.5.2. För att beräkna denna behövdes värden såsom flödeshastighetsgradienten vilket inte fanns tillgängliga i koden, dessa behövde därför tas in från den viskösa flödesdelen av koden. Genom forsatt studie av litteratur som sammanställts i 2.5.2 fick gruppen större förståelse av Ducros-sensorns funktion och hur den implementeras. Först beräknas Ducros-koefficienten Φ antingen på cellväggen eller i cellerna (mer om detta i 3.2.1). Denna multipliceras sedan med Jameson-koefficienten Ψ från ekvation 2.26 och som det kan ses beräknas Ψ både i cell (i, j) och (i+1, j) i fluxmolekylen. Ducros-koefficienten Φ multipliceras sedan med Jameson-koefficienterna Ψ_i och Ψ_{i+1} och slutlugen väljs det största värdet utav produkterna för att få fram Ducros-sensorn

2.27.

3.1.3 Implementering av nya sensorn i koden

Då formeln som används är definierad var implementationen i koden relativt enkel. Först beräknades hastighetsgradienterna på cellväggen i varje fluxmolekyl och dess värden sparades i *numpy arrays* där varje element representerar en cell. Med dessa arrays beräknades divergensen och rotationen av hastigheten som sedan sparas i ytterligare arrays som används för att beräkna Ducros koefficienten Φ. Ducros-sensorn beräknas genom att ta produkten av Φ vektorn och Jameson koefficient-vektorn Ψ från båda sidor väggen. Sedan tas det största värdet av de två elementen i varje cell för att få en vektor av filtreringskonstanter. Så som koden såg ut tidigare fanns det en if-sats som beräknar Jameson-sensorn om stötdämpning används. För att implementera den nya koden behövdes beräkningen av en sista konstant "cfilt" flyttas ut eftersom denna ska vara Ducros-sensorn i de fall då Ducros används. Koden kan ses i bilaga A.2

En ny if-sats skapades som kontrollerar om Ducros-sensorn är konfigurerad till att användas. I if-satsen beräknades Ducros-sensorn i ett antal steg. Efter denna del kontrollerar koden återigen om fnctp är större än 0 och alltså om dämpning används. Därefter så sätts vektorn av konstanter cfilt, som används senare i programmet, till att vara Ducros eller Jameson-sensorn beroende på vilken som används. Med det så uppnåddes målet med att implementera Ducros-sensorn och programmet gick att köra.

Ett problem som uppstod var att koden inte fungerade då Ducros inte används men att stötdämpning fortfarande används. Det som hände var att koden försökte multiplicera Jameson-sensorn med konstanterna Ducros1 och Ducros2 som då inte finns. För att fixa problemet så sattes dessa konstanter till att vara vektorer med ettor i delen där Jameson-sensorn beräknas vilket gjorde att beräkningen för "cfilt" fungerade oavsett om Ducros används eller ej, om Ducros används skrivs ettorna över med de beräknade värdena. Med detta det också väljas vilken sensor som ska användas (detta är dock mer relevant för detta projektet då flera test utfördes och typen av sensor ofta behövdes bytas).

3.1.4 Simulering av icke-viskösa testfall med Jameson och Ducros-sensor

För att testa skillnaden mellan den gamla och nya metoden kördes testfallen: "Ramp", "Q1DShockTube" och "Q1DNozzle". Dessa är beskrivna i sektion 2.6. För alla dessa fall kördes koden både med endast Jameson-sensorn och med Ducros-sensorn konfigurerad. Förutom denna skillnad var testerna i övrigt konfigurerade och körda på exakt samma sätt. Testerna genererade figurer och resultat som sparades, se kap 4.1. Som del av arbetsgången analyserades dessutom dessa resultat för att då validera att Ducros-sensorn funkar som den ska och att projektet kunde ta sig till fas två.

3.2 Stöthantering för viskösa flöden

Under projektets gång uppdaterades "G3DFlow.py" med funktionalitet för att köra simuleringar med visköst flöde. Förändringen ändrade inte något med implementationen av Ducros-sensorn då denna fungerade på samma sätt ändå. I denna del undersöktes skillnaden mellan Jameson och Ducros-sensorn då Ducros-sensorn bör prestera bättre i viskösa flöden. Det undersöktes också vilka skillnader som fanns mellan två alternativa beräknings metoder för att ta fram hastighetsgradienten Δu, det vill säga att gradienten beräknades i cellerna kontra på cellväggen. Notera speciellt att gradienten i Ducros sensorn beräknas vanligtvis i varje *cell* men det blev en del av projektets undersökning att se vad det blev för skillnad om gradienten beräknas mellan två celler ("face-gradient") för att få svar på om det ökade simuleringens prestanda, vad gällande exempelvis beräkningskostnad.

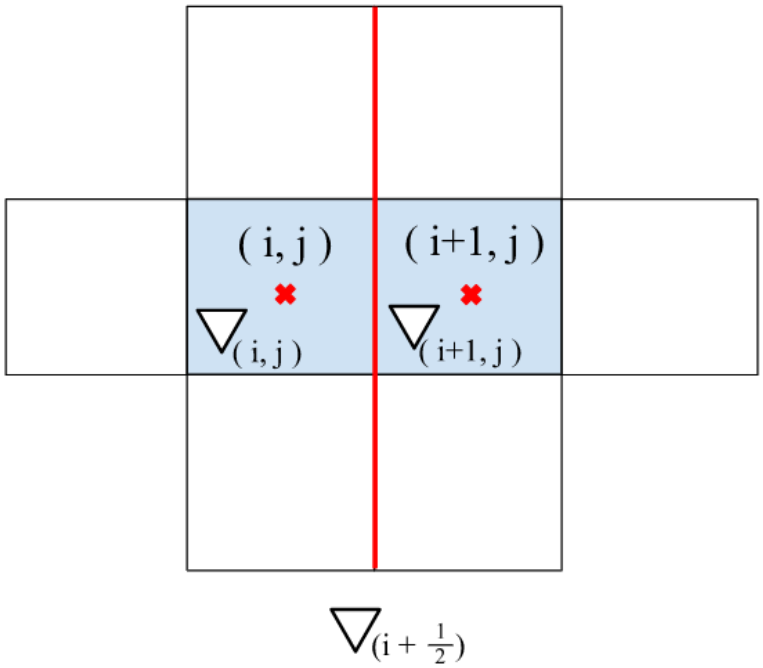
3.2.1 Alternativa Ducros konfigurationer

När Ducros koefficienten Φ beräknas används divergensen och rotationen (eng. curl) av hastigheten, $\nabla \cdot U$ respektive $\nabla \times U$ och denna gradient ∇ kan antingen beräknas i varje cell eller i cellväggen mellan (i, j) och (i+1, j). Men beroende på vilken metod som används för att beräkna hastighetsgradienter ser dessa olika ut. I fallet där hastighetsgradient beräknas på cellväggen fås en vektor med gradienter över alla olika fluxmolekyler, en divergensvektor och en rotationsvektor. När Φ beräknas fås endast ett värde som sedan multipliceras med de båda värdena på Ψ från Jameson-sensorn innan och efter gränsytan mellan två celler, av vilka det största används för att få Ducros-sensorn. Däremot när hastighetsgradienter beräknas i varje cell fås en hastighetsgradient för varje cell i varje fluxmolekyl. När då divergensen och rotationen skall beräknas fås ett värde för varje cell i varje fluxmolekyl. Detta ger två värden på Φ som multipliceras med motsvarande värde på Ψ, det största av dessa produkterna används för att få fram Ducros-sensorn. Skillnaden illustreras i figur 3.1 där gradienten antingen beräknas i cellen: ∇_i och ∇_{i+1}, eller i cellväggen: $\nabla_{i+\frac{1}{2}}$. I ekvation 3.1 och 3.2 förtydligas hur varje variant of Ducros-sensorn fungerar och hur Ducros koefficienten är proportionerlig med gradienten i cellen respektive väggen.

Tillvägagångssättet för att beräkna Ducros-sensorn är olika beroende på vilken hastighetsgradient som används men resultatet blir i slutändan samma sensor. Det som ska undersökas genom test är skillnaden mellan dessa både i beräkningstid och resultat.

$$\texttt{Ducros 1} : \Psi_{i+\frac{1}{2},j}\Phi_{i+\frac{1}{2},j} = max(\Psi_{i,j}\Phi_{i,j}, \Psi_{i+1,j}\Phi_{i+1,j}), \quad [\Phi_{i,j} \propto \nabla_{i,j}, \Phi_{i+1,j} \propto \nabla_{i+1,j}] \tag{3.1}$$

$$\texttt{Ducros 2} : \Psi_{i+\frac{1}{2},j}\Phi_{i+\frac{1}{2},j} = \Phi_{i+\frac{1}{2},j}max(\Psi_i, \Psi_{i+1,j}), \quad [\Phi_{i+\frac{1}{2},j} \propto \nabla_{i+\frac{1}{2},j}] \tag{3.2}$$



Figur 3.1: Metoder för beräkning av hastighetsgradienter

3.2.2 Implementering av Ducros-sensor beräknad med hastighetsgradienter i varje cell

För att kunna välja vilken typ av sensor som ska användas specificerades detta i konfigurationen till de olika testfallen genom att sätta en variabel till ett visst värde. I koden där själva beräkningen görs kontrolleras det valda värdet genom användning av if-satser och sedan utförs därefter olika delar av koden, vilket gjorde att implementationen för beräkning av gradienter över cellväggen kunde behållas och endast behövdes läggas in i en if-sats. En liknande if-sats kontrollerar sedan om gradienten i cellytans centrum ska beräknas om denna variant av Ducros-sensorn ska användas.

Om Ducros-sensorn med cellbaserade gradienter är vald beräknas hastighetsgradienterna i varje cell. Gradienterna används sedan för att få fram rotation och divergens i cell 2 och 3 i varje fluxmolekyl varefter de sedan användes för att få fram värden på Φ i cell 2 och 3. Dessa multipliceras sedan in med respektive värde på Ψ i cell 2 och 3 från Jameson-sensorn och maximum av produkten innan och efter cellväggen används för att få Ducros-sensorn med gradienten beräknad i varje cellcentrum.

3.2.3 Konfigurering och körning av visköst testfall

Testfallet är det som är beskrivet i sektion 2.6.7. Detta fanns färdiguppsatt i koden men behövdes konfigureras för att köras med de olika metoderna för shock capturing. Testet utfördes med Jameson-sensorn och Ducros-sensorn där hastighetsgradienten beräknades i cellväggen för ett test och i mitten av varje cell

i ett annat. Med andra ord kördes 3 simuleringar för att jämföra det fysiska resultatet mellan Jameson-sensorn och Ducros-sensorn, samt overhead skillnader för beräkningen av gradienten i väggen jämfört med beräkningar i mitten av cellen. Enda skillnaden mellan dessa är värdet som sätts på variabeln *set_shock_dampening_mode* som sätts till JAMESON, DUCROS_FACE_ GRADIENT eller DUCROS_CELL_GRADIENT beroende på vilken beräkningsmetod som ska användas.

För det viskösa testfallet utfördes 20 000 iterationer med "Local time-stepping" och 10 000 iterationer med "Time accurate (CFL)" tidssteg som beskrivet i 2.6.7. Anledningen till detta var på grund av ett antal svårigheter med att utföra tested helt och hållet med CFL som ledde till instabilitet i simuleringen. Med denna konfigurering lyckades lösningen konvergera och visa rimliga resultat som diskuteras i kapitel 4.2.

4
Resultat

Under detta kapitel presenteras resultaten som togs fram före och efter implementeringen av den nya Ducros-sensorn. I sektion 4.1 beskrivs resultaten från simuleringen av de tre icke-viskösa testfallen, nämligen ett kvasi-endimensionellt flöde genom ett munstycke (nozzle flow 2.6.3), ett kvasi-endimensionellt flöde genom ett stötrör (shocktube 2.6.5) och ett tvådimensionellt fall över en ramp (2.6.1). För alla dessa fall förväntas det att en stöt kommer bildas någonstans i flödet. Detta behandlades först av Jameson sensorn och sedan Ducros-sensorn men för dessa icke-viskösa testfall skall implementeringen av Ducros-sensorn *inte* påverka resultatet. Det eftersom testfallen redan har validerats att vara korrekt före den nya implementeringen och eftersom Ducros-sensorn förebygger problem för viskösa flöden bör den inte påverka resultat för simulering av icke-viskösa testfall.

I sektion 4.2 utfördes ett testfall med visköst flöde där en stöt interagerar med ett gränsskikt (2.6.7) som får flödet att virvla och i detta fall förväntas Ducros-sensorn att påverka numeriska resultaten.

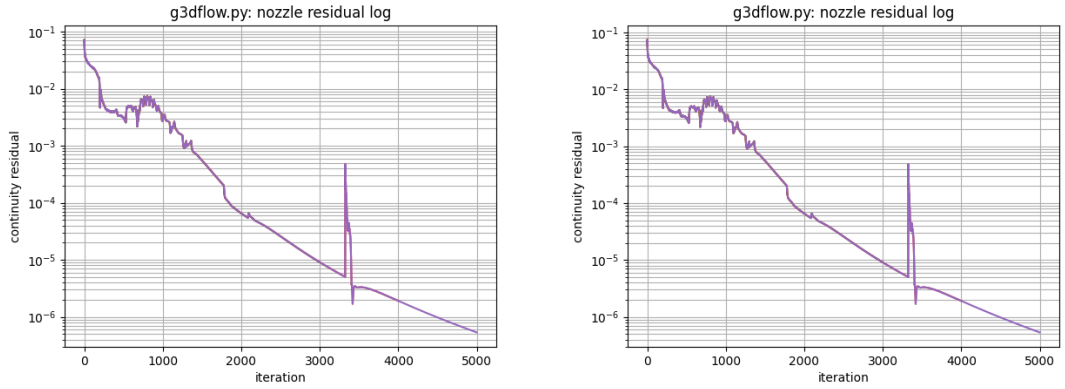
I sektion 4.3 presenteras resultaten av olika beräkningskonstnader beroende på hur Ducros-sensorn implementeras och jämförs med originella Jameson-sensorn. Som beskrivet i metodikdelen implementeras Ducros-sensorn med två metoder: antingen beräknas gradienten mellan två celler eller inom varje cell. Givet att numeriska resultatet är detsamma i båda fall och ger jämförbara fysikaliska konsekvenser jämförs beräkningskostnaden så att en av dessa metoder kan sållas bort. Att Ducros-sensorns beräkningskostnad jämförs med Jameson sensorn är relevant för att få en förståelse av konsekvensen av den nya implementeringen i koden. Det är dock inte något som används för att göra beslut om att använde Ducros-sensorn eller inte, i och med att den *bör* användas då flödet är visköst.

4.1 Resultat från implementation av Ducros-sensor i icke-viskösa flöden

Vid jämförelse av resultaten före och efter implementeringen av Ducros-sensorn användes en jämförelse mellan kvarstående felet (*residual*) som funktion av antalet iterationer för att se om numeriska lösningen konvergerar lika snabbt i båda fallen. Sedan utfördes en jämförelse av flödesegenskaper över en horisontell mätningssond i x-led, i detta fall densitet, tryck, temperatur, Mach-tal, totala trycket och totala temperaturen.

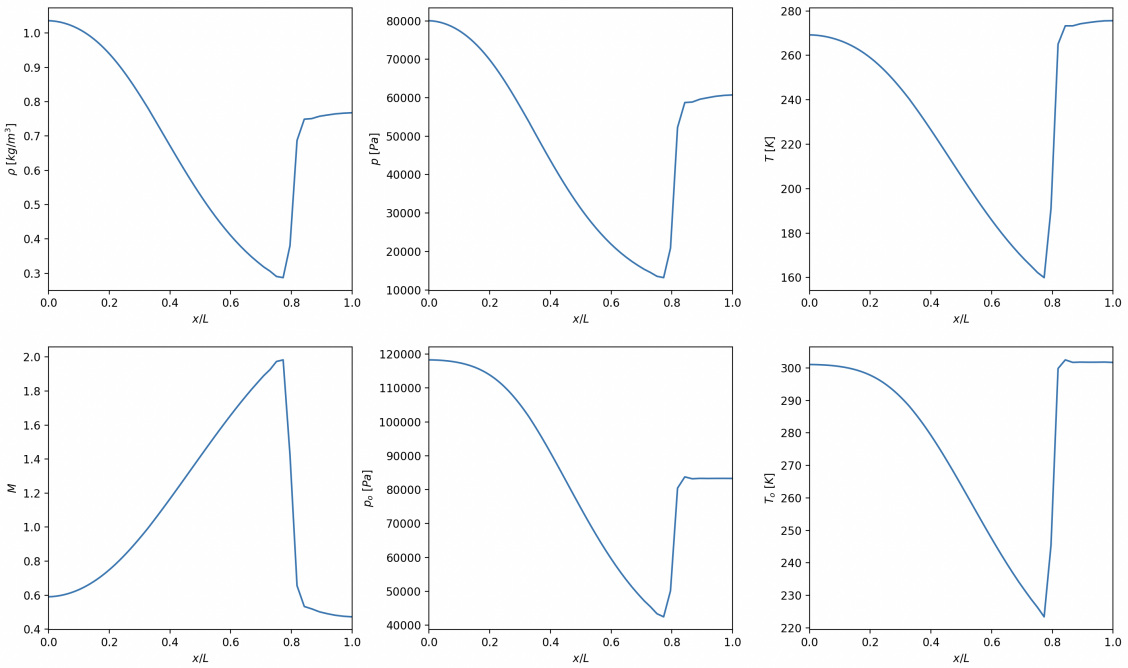
4.1.1 Q1D-nozzle resultat

I figurerna 4.2 och 4.3 utfördes numeriska lösningen utan Ducros-sensorn och sedan med Ducros sensorn för ett konvergerande-divergerande munstyck. I figur 4.1 plottades för beräkningen genomförd med Jameson och beräkningen genomförd med Ducros och i både fallen ses att efter ungefär 5000 lösningsiterationer är det kvarstående felet av godtagbar storlek. Grafernas utformning är dessutom nästan identiska vilket pekar på att lösningen konvergerade likadant i båda fallen.

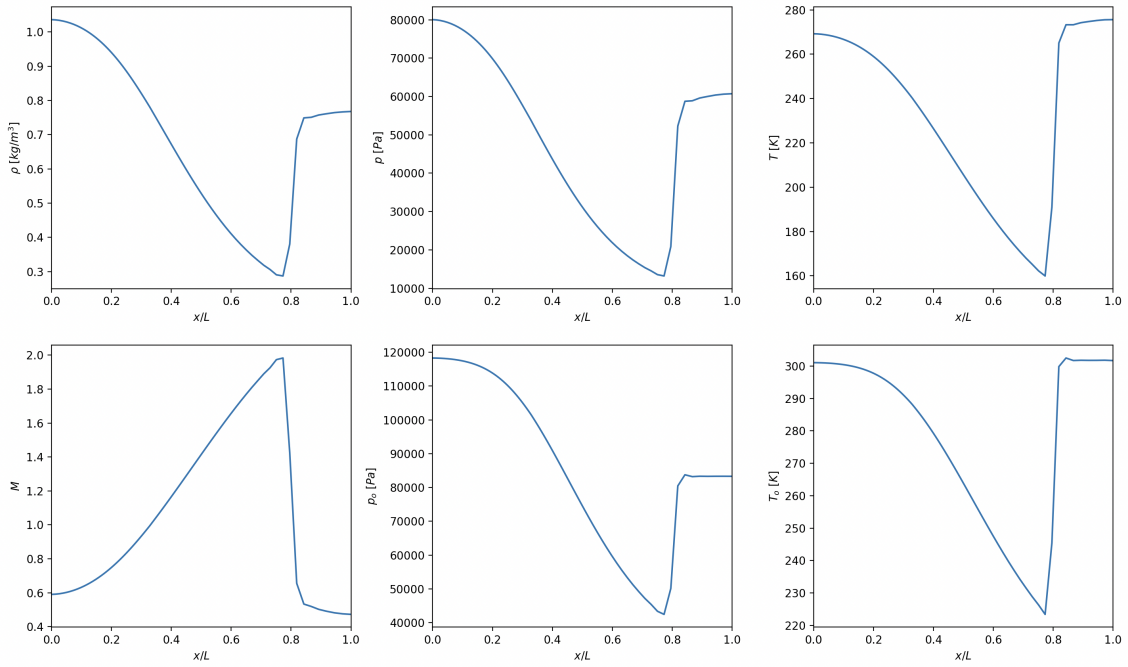


Figur 4.1: Kvarstående felet för Jameson- respektive Ducross-sensorn som funktion av antal iterationer för ett kvasi-endimensionellt konvergerande-divergerande munstycke

I figurerna 4.2 och 4.3 plottas flödesegenskapernas variation i x-led. Det som märks mest tydligt med alla flödesegenskaper är den stora gradienten som sker runt 0.75 m. Detta är förstås stöten som orsakar drastiska ändringar av flödesegenskaperna. I panelen i andra raden, första kolumnen ses både för Jameson- och Ducros-sensorn att Mach-talet stiger till ungefär 2.0 och sedan snabbt avtar ner till en subsonisk nivå (ungefär 0.5) vilket stämmer överens med termodynamikens andra lag (det kan visas att för att entropi ska öka över en stöt måste flödet bli subsoniskt efter stöten [5][s.92]). Det som igen är viktigt att notera är att implementeringen av Ducros-sensorn inte har ändrat numeriska lösningens förmåga att uppskatta de fysikaliska egenskaperna av flödet och detta är konsekvent med alla mätningar av flödesstorheter.



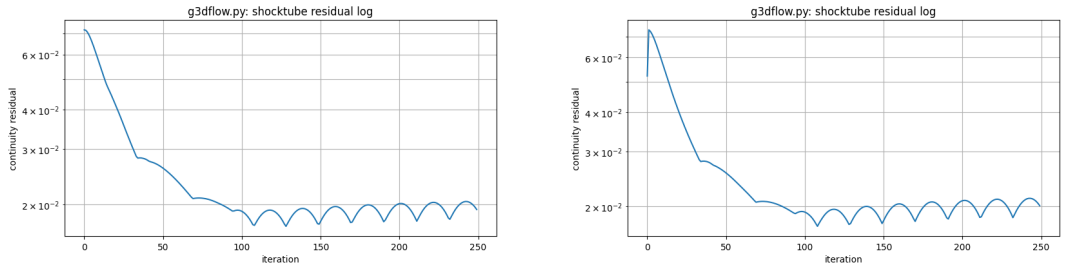
Figur 4.2: Flödesegenskaper för ett kvasi-endimensionellt konvergerande-divergerande munstycke med Jameson-sensorn



Figur 4.3: Flödesegenskaper för ett kvasi-endimensionellt konvergerande-divergerande munstycke med Ducros-sensorn

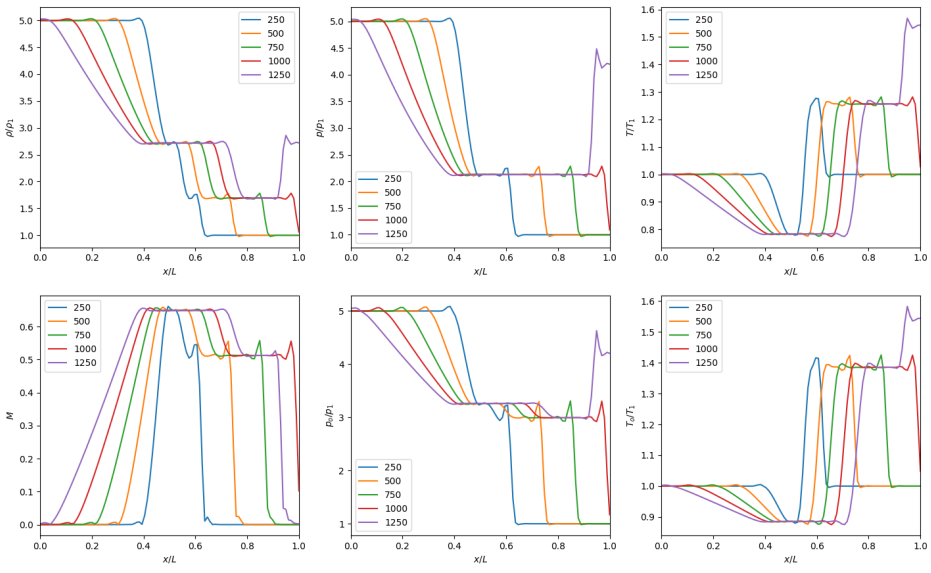
4.1.2 Q1D-shocktube resultat

I figur 4.4 plottas kvarstående felet för stötröret och det som skiljer sig från konvergerande-divergerande munstycket är att simuleringen har stannat runt $2 * 10^{-2}$ och det har därmed slutat mäta efter 250 iterationer vilket är avsiktligt, som förklarat i 2.6.5. Vidare itererar koden med steg på 250 och plottar flödeskonstanterna för varje sådant steg vilket kan ses i figur 4.6 och 4.5.

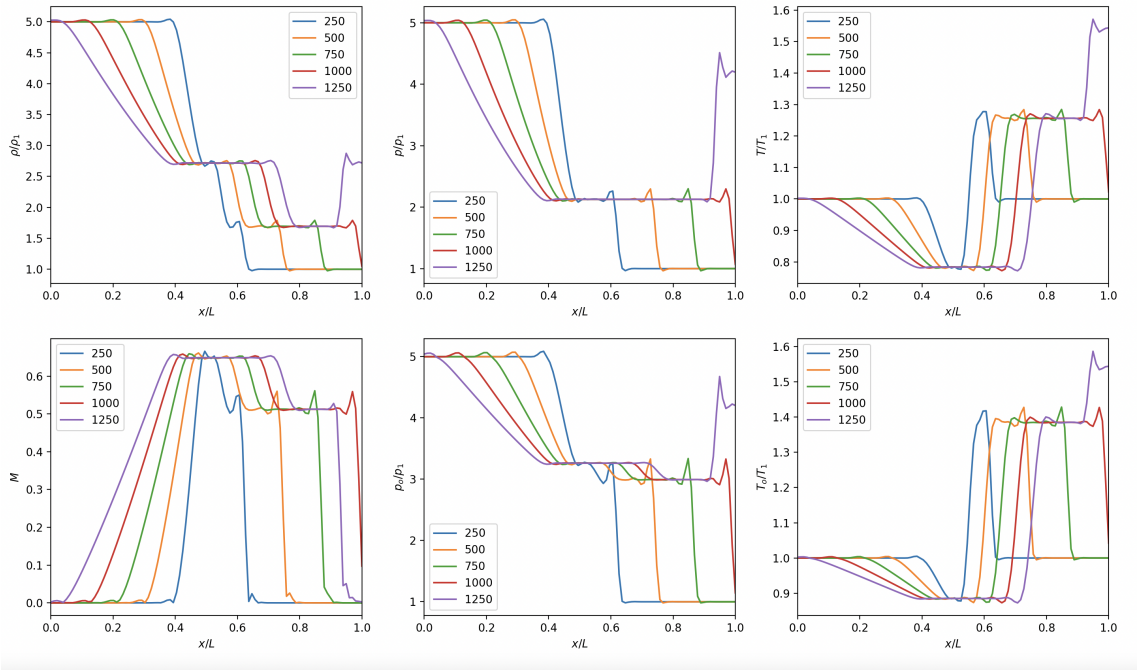


Figur 4.4: Kvarstående felet som funktion av antal iterationer för ett kvasi-endimensionellt stötrör

I figur 4.4 noteras även att båda lösningars kvarstående fel som funktion av antalet iterationer är mycket lika även om felet för Ducros-sensorn börjar på en lägre nivå.



Figur 4.5: Flödesegenskaper för ett kvasi-endimensionellt stötrör med Jamesonsensorn

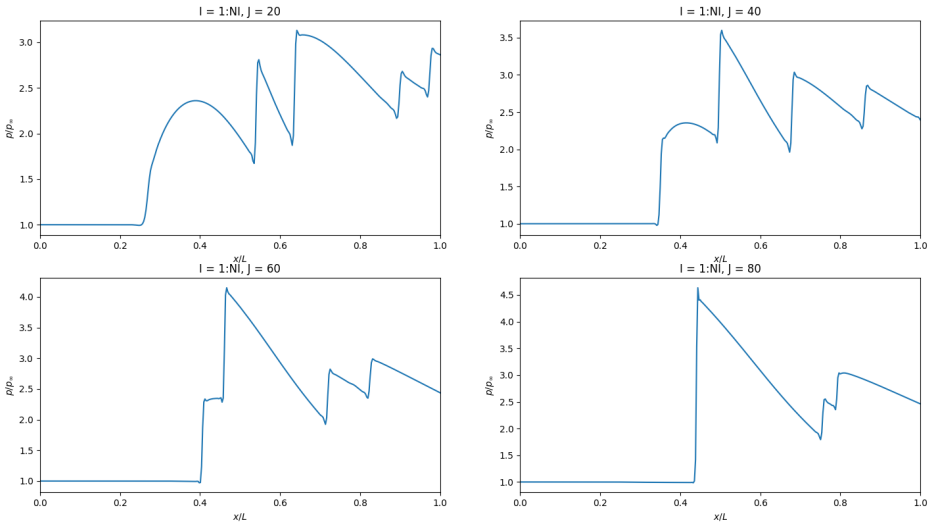


Figur 4.6: Flödesegenskaper för ett kvasi-endimensionellt stötrör med Ducros sensorn

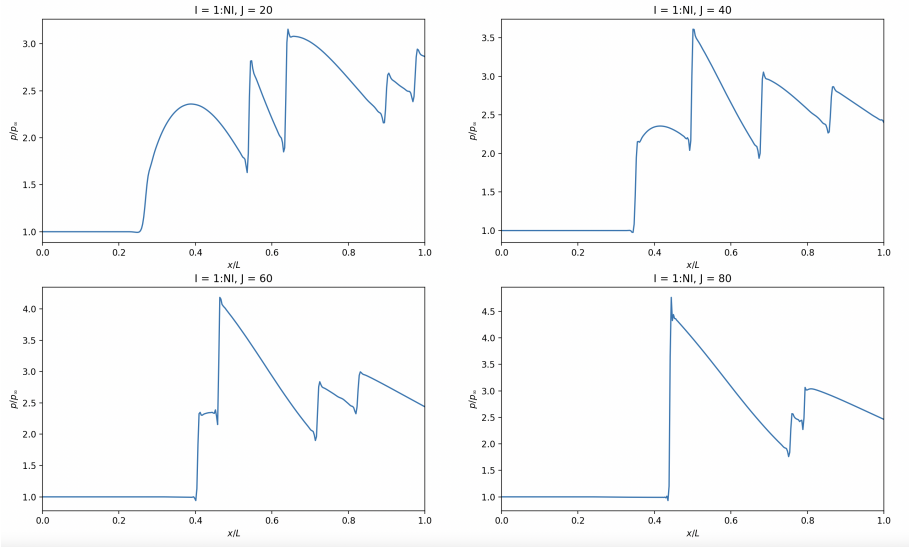
I figur 4.5 och 4.6 visas olika flödeskonstanters variation i x-led över stötröret, de olika färgerna symboliserar olika antal iterationer och är därför olika tidpunkter i simulationen. Desto fler iterationer, desto längre har simuleringen körts, det vill säga, desto längre tid för flödet att utvecklas och det noteras att stöten rör sig genom röret över tid. När figurerna 4.5 och 4.6 jämförs är det svårt att se någon skillnad mellan resultaten från de två simuleringarna.

4.1.3 Ramp resultat

I figur 4.7 och 4.8 visas tryckets variation i x-led i olika höjder från ramp simuleringen med Jameson och Ducros-sensor. I figur 4.9 ställs det visualiserade resultatet från de två simuleringarna upp mot varandra för jämförelse, där Machtal och tryck visas.

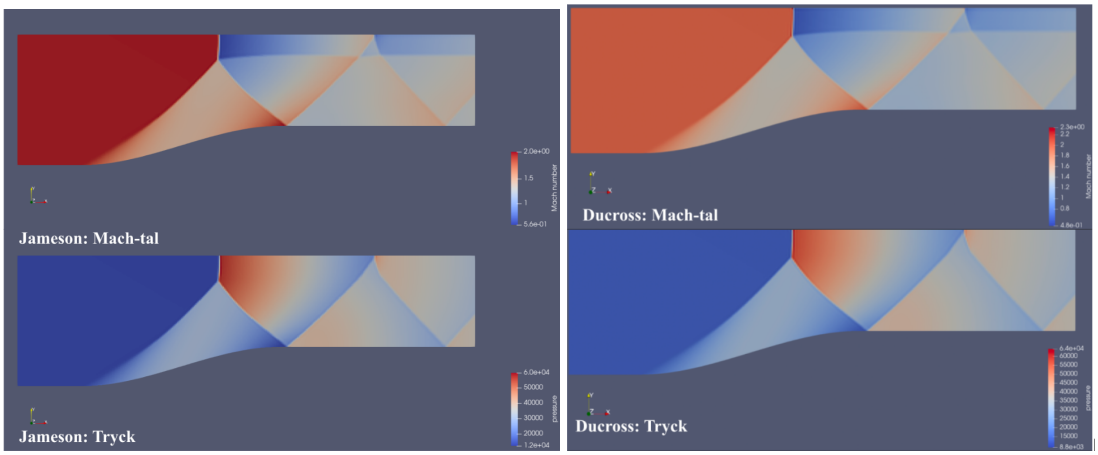


Figur 4.7: Flödesegenskper för ramp testfallet med Jameson-sensorn



Figur 4.8: Flödesegenskper för ramp testfallet med Ducros-sensorn

Om figur 4.7 och 4.8 jämförs är det svårt att se någon betydelseful skillnad mellan dom och resultaten ser mycket lika ut.



Figur 4.9: Ramp diagram med Mach-tal och tryck

Om resultaten från simuleringen jämförs i figur 4.9 så kan vissa skillnader i framförallt Mach talet noteras. Simuleringen med Jameson-sensorn är mer mörkröd i färgen till skillnad från den med Ducros-sensor vilket beror på att skalan är annorlunda. Om de faktiskta värdena jämförs är det egentligen inte någon större skillnad mellan de två simulerings resultaten. Det samma gäller för visualiseringarna som visar trycket.

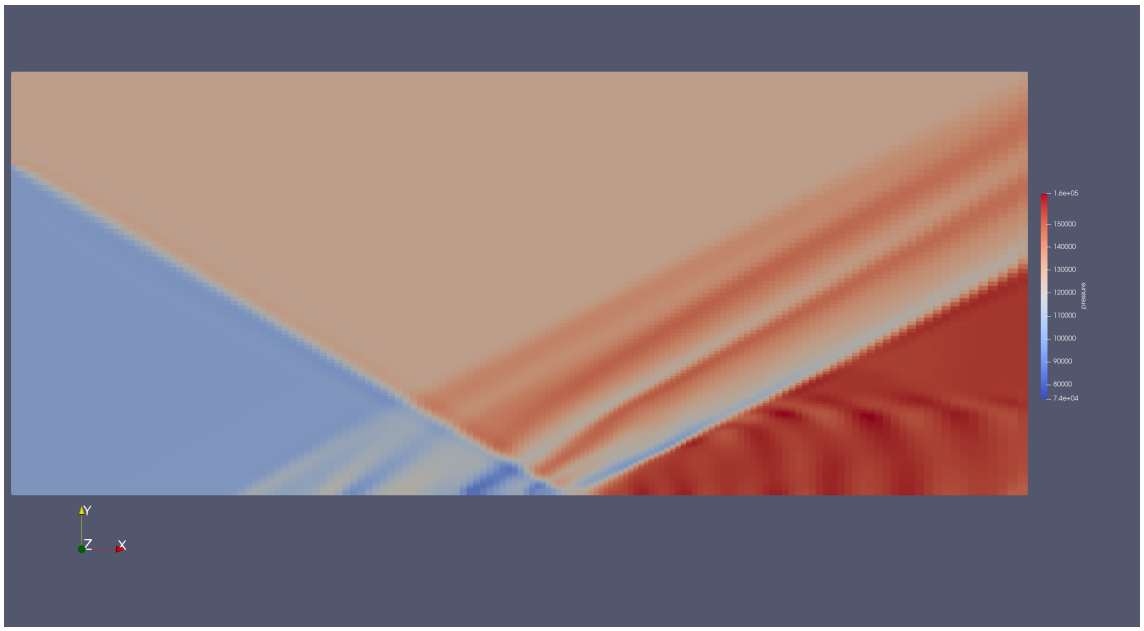
4.2 Resultat från visköst flöde med Ducros-sensor

Nedan presenteras resultaten som togs fram med Jameson och Ducros-sensorerna efter implementeringen av viskösa flöden för testfallet *visköst laminärt flöde över en platta*. Notera att tidsstegen inte stämmer överens mellan testen, förutom i 4.17.

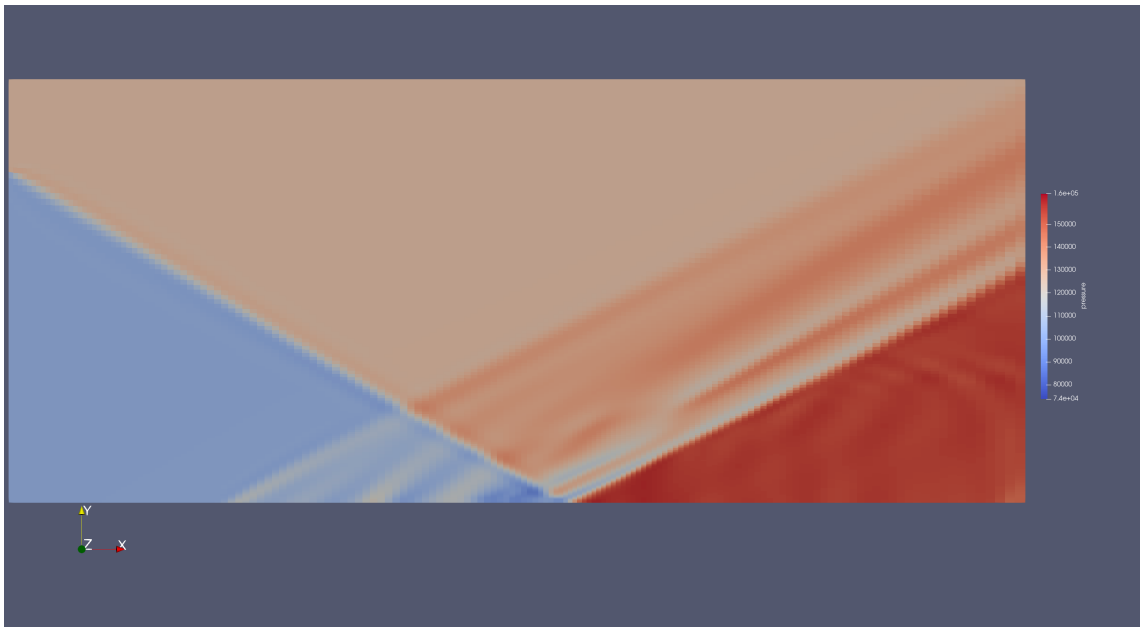
Figur 4.10: Mach-tal med Jameson-stöthantering

Figur 4.11: Mach-tal med Ducros-stöthantering, med gradienter beräknade i celler

När figurerna 4.10 och 4.11 jämförs så noteras att de är väldigt lika, men med vissa små skillnader. Bland annat är gränsskiktet något tunnare i fallet med Ducros-sensorn i jämförelse med Jameson. I fallet med Jameson-sensorn noteras även att det finns några små förändringar i hastighet i det nedre högra hörnet som inte förekommer i simuleringen med Ducros-sensorn.



Figur 4.12: Tryck för simulering med Jameson-stöthantering



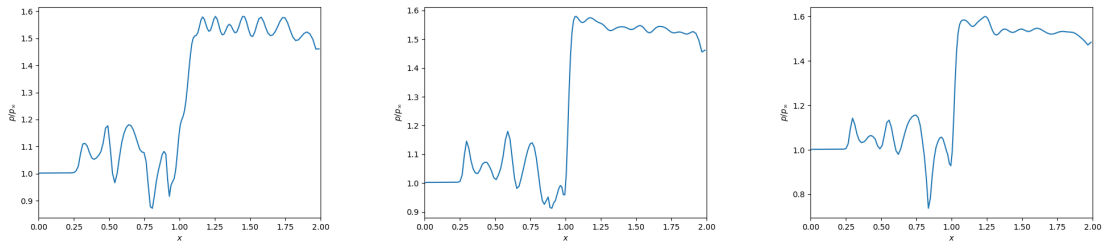
Figur 4.13: Tryck för simulering med Ducros-stöthantering med gradienter beräknade i celler

Figurerena 4.12 och 4.13 är även de mycket lika, men visar några små skillnader. En är att det är ett mer ojämnt tryck i regionen i nedre högra hörnet på simuleringen med Jameson-sensorn, vilket inte förekommer i simuleringen med Ducros-sensorn som istället visar ett jämnare tryck i denna region. Ytterligare visas även skillnad i vågorna som orsakas av att flödet träffar framsidan av plattan, i simuleringen med Jameson-sensorn är det större gradienter i dessa till skillnad från Ducros-sensorn där det är mer som ett fält av högre tryck än omgivningen.

Figur 4.14: Mach-tal med Ducros-stöthantering med gradienter beräknade på cellväggen

Figur 4.15: Tryck för simulering med Ducros-stöthantering och gradienter beräknade på cellväggen

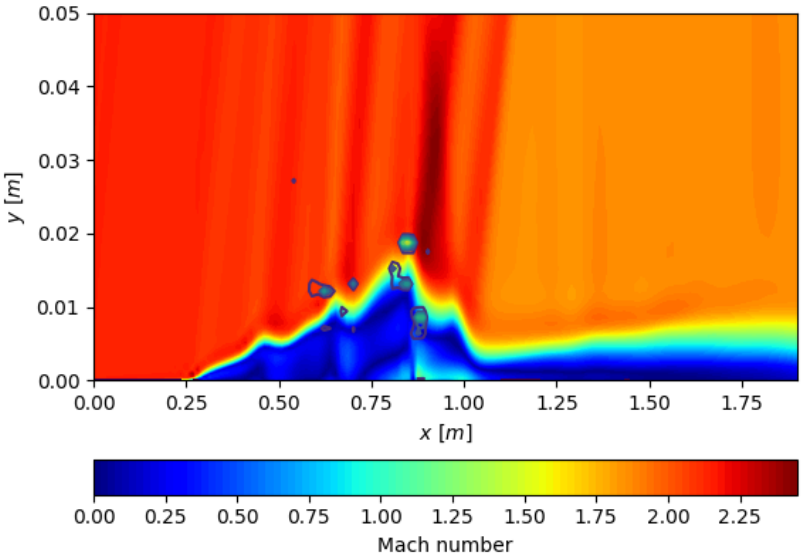
Jämförs simuleringen med Ducros-sensor som använder gradienter beräknade på cellväggen 4.14 4.15 med den som använder gradienter beräknade i cellerna 4.11, 4.13 så noteras vissa skillnader, speciellt i figurerna som visar tryck. Skillnaderna är exempelvis mindre tydliga vågor i 4.15 i jämförelse med 4.13 där dom är skarpare med större gradienter. Förutom detta visas även ett mycket lägre tryck innan stötens reflektionspunkt i 4.15 i jämförelse med 4.13.



(a) Jameson Probe **(b)** Ducros cell Probe **(c)** Ducros face Probe

Figur 4.16: Resultat från simulering med sensorer. Mätningarna är tagna 7 celler upp från botten och mäter tryck över stagnationstryck

I figur 4.16 så noteras några skillnader mellan de olika simuleringarna. Om delen efter stöten observeras noteras att 4.16a är mindre stabil och går mer upp och ner i jämförelse med 4.16b och 4.16c där det är mer stabilt. Vidare noteras också att både 4.16c och 4.16b har en större gradient där stöten inträffar i jämförelse med 4.16a. Om området precis innan stötenobserveras är 4.16a och 4.16c ganska lika medan 4.16b har ett stabilare lägre tryck precis innan stöten.



Figur 4.17: Jämförelse mellan Jameson och Ducros i gränsskiktet

Den verkligt intressanta jämförelsen är hur de två sensorerna skiljer sig på mikronivå. Figur 4.17 visar mach-tal i ett tidssteg för en simulering med Ducros. Ovan Ducros-simuleringen visas regioner inringade med blått där Jameson-sensorn beräknar en högre dämpningskoefficient, men som Ducros-sensorn inte dämpar. Notera att detta representerar simuleringsresultat i samma tidpunk och är därför direkt jämförbara

4.3 Jämförelse av overhead

För att se hur mycket extra beräkningskostnad som tillförts av de olika versionerna av Ducros-sensorn så kördes simuleringar ett antal gånger där körtiden uppmättes. Av körtiderna togs sedan ett medelvärde och en standardavvikelse för att kunna jämföra dem. Testet gjordes på det viskösa testfallet 2.6.7 där testet kördes ett antal gånger med 1000 iterationer och med *Local Time Stepping* med de olika sensorerna.

Tabell 4.1: Resultat från test av beräkningskostnad med visköst flöde

Metod	**Medelvärde (s)**	**Standardavvikelse**
Jameson	10.8966	0.05448210715455146
Ducros face	12.4280	0.03326409475695987
Ducros cell	12.5936	0.03393081195609703

I testet visas att Jamson sensorn gav klart snabbast körtid med en standardavvikelse som är lägre än skillnaden upp till nästa sensor. Ducros sensorerna var väldigt lika i körtid men ändå större skillnad mellan de olika versionerna än dess standaradavvikelse. Trotts att det är liten skillnad så är Ducros-sensorn med gradienter beräknade i cellerna långsammare än Ducros-sensorn med gradienter beräknade på cellväggen.

5
Diskussion

Följande kapitel beskriver de slutssatser som tagits genom att tolka resultatet i kap 4 och jämföra det med förväntade utfallen i kap 2. Om ett resultat avvikit från ett förväntat resultat har det noterats.

5.1 Implementeringen av Ducros utan viskositet

Från resultaten i kap 4.1 kan tydas att Ducros-sensorn inte påverkar simuleringen över det icke-viskösa flödesfallen vilket är precis det resultat som är önskat. Att sensorn inte påverkar gäller för alla de testfall som utförts (munstycke, stötrör och ramp). Figur 4.1 visar hur många lösningsiterationer som var nödvändigt för att uppnå ett kvarstående fel av godtagbar storlek där simuleringarna med Jameson och Ducros gav nästintill identiska resultat, vilket ytterligare underbygger att ändringen av sensor inte påverkar negativt. Detta var fallet även för kvarstående felet för samtliga resterande testfall. Som nämnt i resultatdelen är kvarstående felet inte riktigt ett mått på hur fysikaliskt realistiskt resultaten är, men att konvergensen för de olika sensorerna är snarlik tyder på att de konvergerar lika snabbt och är lika stabila simuleringar. Efter att det konstaterats att simuleringar med de båda sensorerna konvergerar mot samma sak kan det nu göras en tolkning av hur realistiskt simuleringen beter sig för båda sensorerna.

I kapitlet där de olika testfallen beskrivs (2.6) förklaras också vilka resultat som är förväntade utifrån existerande litteratur och experiment. Som nämnt plottades flödesstorheterna densitet, tryck, mach-tal och temperatur för munstycket i grafer och det var tydligt från figur 4.2 och 4.3 att de olika sensorerna inte skiljer sig åt. Notera att för Jameson-sensorn är detta resultat sedan länge redan validerat och i och med att Ducros-sensorn inte påverkade resultatet implicerar detta förstås att Ducros-sensorn också har producerat korrekta resultat. Det är dock värt att nämna att simuleringen i båda fall faktiskt beter sig som förväntat, det vill säga, att Machtalet (och därmed hastigheten av flödet) stadigt ökar tills den når stöten och därefter minskar betydligt och diskontinuerligt. För de andra flödesegenskaperna kan observeras att de stadigt minskar då hastigheten ökar (detta är dessutom en mycket fundamental konsekvens av Eulerekvationerna) tills det att stöten kraftigt bromsar upp flödet. Inbromsningen gör att trycket, densiteten och temperaturen ökar, något som är väntat från teorin [8, s. 623] [5, s. 205].

För stötröret visade sig resultaten för Jameson-sensorn och Ducros-sensorn vara

näst intill identiska. Det var sant både för det kvarstående felet och jämförelsen av flödesegenskaperna för olika iterationer. Något som noteras är att alla flödesegenskaper för alla iterationer hade samma utformning vilket pekar på att stöten i stötröret rör sig genom röret på exakt samma sätt både för Jameson-sensorn och Ducros-sensorn. Jämförs det fysiska resultat kortfattat med det förväntade resultaten utifrån experimentell data från kapitel 2.6.5 kan det ses att simuleringen beter sig som förväntat. Alltså är tryck (och densitet och temperatur) högt i den drivande delen av röret och minskar sedan i diskontinuerliga "trappsteg" på grund av expansionsvågorna och normalstöten. Slutligen är det som förväntat att de diskontinuerliga gradienterna (som visar var normalstöten är) rör sig i x-led mot den drivande delen då antalet iterationer ökar och detta är analogt med en stöt som rör sig längs röret vilket också är fysiskt förväntat.

Som för munstycket och stötröret är simuleringen av rampen den samma för Ducros- och Jameson-sensorn och med detta konstateras det att alla testfall har pekat på att Ducros-sensorn inte försämrar det fysiska resultatet. Jämförs det fysiska resultatet från kapitel 2.6.1 noteras att, som förväntat, det bildas en stöt vid geometriändringen (där rampen sluttar uppåt) som reflekteras mot *taket* av simuleringen. Med det sagt kan dras slutsatsen dras att, utifrån att Ducros-sensorn ger identiska resultat som Jameson-sensorn och att alla resultat stämmer överens med teori och experimentell data, att Ducros-sensorn korrekt lyckas identifiera och hantera stötar.

5.2 Implementeringen av Ducros med viskositet

Inledningsvis kan vid okulär besiktning av figurer som visar Machtal 4.10, 4.11 och 4.14 samt figurer som visar tryck 4.12, 4.13 och 4.15 konstateras att ur ett makro-perspektiv är de näst intill identiska. Likheten tyder på att ändringen av sensor inte förändrar beteendet allt för mycket, vilket är önskvärt då Jameson-sensorn redan ger en god approximation utanför gränsskiktet.

Som nämnt skiljer sig tidsstegen mellan simuleringarna åt men då det som är av intresse ändå framkommer, det vill säga hur simuleringarna skiljer sig åt på en makro-nivå, bedöms det som godtagbart. Däremot när skillnaderna i gränsskiktet jämförs i figur 4.17 är resultatet taget i samma tidssteg, vilket gör skillnaderna absoluta och jämförbara.

Jämförs resultatet från kap 4.2 och det förväntade utfallet 2.6.8 visas tydliga liknelser från simuleringen och Schlieren-visualiseringen av ett liknande fall [2, s.252].

Som gestaltat i figur 4.17 presterar ducros-sensorn som förväntat i hänseendet att den inte dämpar alla regioner som jameson-sensorn gör. Det som observeras är att skillnaderna endast uppstår där fluktuationer i hastighet är signifikanta. Enligt teorin ska virvlar förekomma vid knölen på gränsskiktet och det är precis det som kan observeras. Jameson-sensorn kan inte skilja virvlarna från stötar, vilket gör att de blir artificiellt dämpade. Ducros-sensorn kan skilja virvlarna från

stötar, och applicerar därför ingen dämpning på dem vilket gör att mer energi behålls i systemet och blir mer fysikaliskt korrekt. Då skillnaderna uppstår där teorin hävdar att det ska förekomma virvlar, beskrivet i kapitel 2.6.8, kan det med relativt stor säkerhet påstås att för detta specifika testfall gör Ducros-sensorn sitt jobb och innebär en klar förbättring jämfört med Jameson-sensorn.

5.3 Jämförelse av overhead

I tabell 4.1 visas resultatet från tidsmätningen av körtiden med de olika sensorerna. I tabellen visas att Jameson-sensorn är klart snabbare än Ducros-sensorn. Detta är inte helt oväntat då Ducros utför fler beräkningar och tyngre operationer. Först ska hastighetsgradienterna tas fram på något sätt, efter det måste gradienterna användas för att beräkna en Ducros-koefficient som sedan ska multipliceras med Jameson-koefficienten. Med andra ord om Ducros-sensorn ska användas måste hastighetsgradienter och Ducros-koefficient beräknas utöver att beräkna Jameson-koefficient och det är därför inte oväntat att detta ger längre beräkningstid. Skillnaden är trotts detta relativt liten, det tar endast ca 14% längre tid med Ducros-sensorn.

Jämförs skillnaden mellan de olika metoderna för beräkning av hastighetsgradienter som används för att beräkna Ducros-sensorn så är det väldigt lite tidskillnad men ändå betydande. Hastighetsgradienter beräknade i varje cell är långsammare än hastighetsgradienter beräknade på varje cellvägg. Detta kan bero på att det utförs fler beräkningar i fallet med cell gradienter då dessa måste bestämmas i varje cell och sedan krävs det att två Ducros-koefficienter beräknas per cell, en på varje sida. Hypotesen var ändå att cellbaserade gradienterna skulle vara snabbare då dessa är snabbare att ta fram men det visar sig att själva beräkningen av två Ducros-koefficienter tar ut detta och netto resultatet blir långsammare. När hastighetsgradienter beräknade på cellväggen används beräknas alltså endast en gradient per fluxmolekyl och denna används för beräkningen av en Ducros-koefficient och det är troligen denna besparing som leder till kortare beräkningnstid.

Det blir tydligt att användandet av en Ducros-sensor ger längre beräkningstid, trotts detta är det inte en stor ökning men ändå betydelseful. Från det vi sett i simuleringarna med icke-visköst flöde så blir resultatet samma mellan Jameson och Ducros-sensorn och det kan därför finnas en poäng i att inte använda sig av Ducros-sensorn om flödet är icke-visköst för att bespara beräkningstid. Däremot när flöder är visköst så har Ducros-sensorn vissa fördelar som gör den mer tillförlitlig och det är därför rekommenderat att använda den trotts något längre beräkningstid.

6
Slutsats

Sammanfattningsvis kan det konstateras att projektet har besvarat majoriteten av frågeställningarna från 1.3. Även målet att förbättra koden är uppnått, bortsett från att koden tar längre tid att köra eftersom fler beräkningar måste göras. Det har även visats att i ett specifikt testfall identifierar Ducros-sensorn stötar bättre än Jameson-sensorn. I detta fall uppfattar Jameson-sensorn virvlande som en stöt och försöker dämpa den, Ducros-sensorn kan däremot känna av effekter av flödesrotation och särskilja det från en stöt och tillämpar därför ingen dämpning. Detta leder till mer tillförlitligt och verkligt resultat. För att kunna säga att detta gäller mer generellt hade fler testfall behövts undersökas för att validera resultatet.

Om projektet hade fortgått hade nästa steg varit att se över implementationen av turbulens i koden samt utföra fler och större simuleringar med visköst flöde för att försäkra att lösningen fungerar för alla geometrier och flödesfall samt erbjuder förbättring gentemot Jameson-sensorn. Detta tillsammans med experiment för att undersöka hur simuleringarna stämmer överens med verkligheten. Mer specifikt hävdar Ducros själv [4, s.524] att för mycket dämpning i systemet leder till konsekvenser längre ner i flödet och att de fysikaliska felaktigheterna därmed blir mer inkorrekt. Med detta sagt hade ett nytt testfall introducerat mycket mer virvlande och ett längre gränsskikt (en flödesdomän som är längre i x-led) så att det tydligt ses att Jameson-sensorn introducerar för mycket energiförluster jämfört med Ducros-sensorn på grund av felidentifiering av stötar. Med det skulle även fler uppgifter kunna framställas och användas i utbildande syfte.

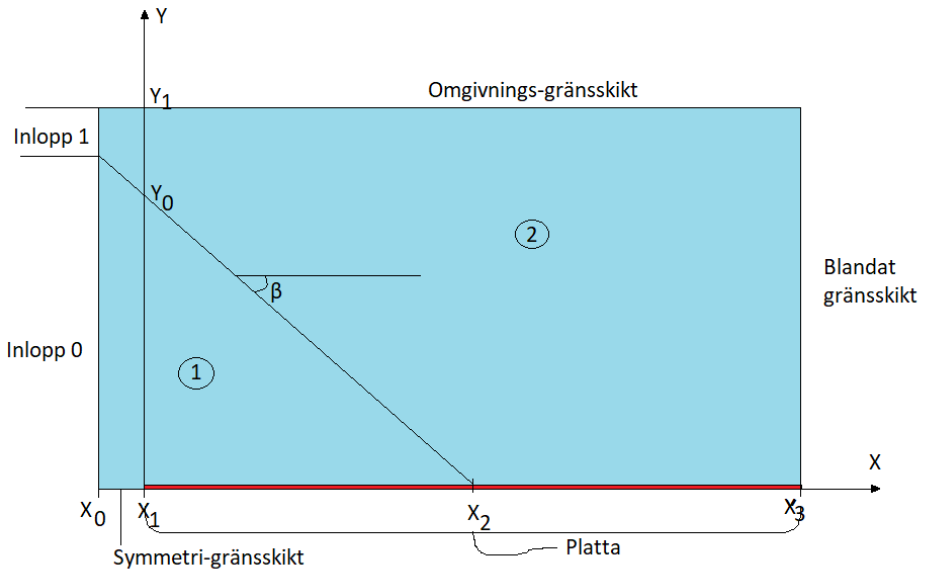
7

Projekt-Case till TME085

Utöver arbetet som gjordes med koden och simuleringarna mynnade projektet som sagt även ut i ett projekt-case som skall användas i kursen TME085-*COMPRESSIBLE FLOW*. Tanken var att resultatet av simuleringar som körts ska ligga som grund för att utforma en uppgift för studenter i kursen. Förbättringarna som genomförts i koden leder till ett mer verkligt resultat av simuleringarna, vilket gör att dessa kan användas som underlag till uppgiften och för studenterna att jämföra sina simuleringar med. Uppgiften är tänkt att användas för att öka studenternas förståelse för CFD-simulering med kommersiell mjukvara och hur komprimerbart flöde beter sig i olika test-fall.

7.1 Genomförande

Till att börja med så gjordes en tydlig definition av hur testfallet konfigureras innan körning där geometri, randvillkor och värdet på flödeskonstanter bestäms. Allt detta behöver vara tydligt beskrivet för att studenterna ska kunna återskapa testfallet på egen hand i annan mjukvara. För att underlätta skapades en figur som beskriver geometrin och de olika flödeszonerna och ränderna, denna kan ses i figur 7.1. Även en tabell på flödeskonstanterna på de olika ränderna kan ses i tabell 7.1 samt längd på delar av geometrin skapades som kan ses i tabell 7.2.



Figur 7.1: Geometri och konfigurering för *Laminärt visköst flöde över en platta*

Tabell 7.1: Beskrivning av geometri

Geometri	Värde
β	30.8°
Inlopp [m]	
$y_0 = 1.2\tan(\beta)$	0.7153
y_1	1
Platta [m]	
x_0	-0.2
x_1	0
x_2	1
x_3	2

Tabell 7.2: Värden för flödeskonstanter

Flödeskonstant	Inlopp 0	Inlopp 1	Utlopp
M	2.15	2.01	2.01
T	$288.00K$	$306.87K$	$306.87K$
p	$0.01bar$	$1.26bar$	$1.26bar$
ρ	$1.23kg/m^3$	$1.44kg/m^3$	$1.44kg/m^3$
a	$340.17m/s$	$351.14m/s$	$351.14m/s$

Efter detta skrevs själva uppgiften. I uppgiftsbeskrivningen ges en kort introduktion till uppgiften och projektet, varpå själva testfallet och geometrin beskrivs för att ge studenterna en uppfattning av vad som skall göras och hur de ska konfigurera sin simulering. Som hjälp på traven ges även förslag på sökämnen som

kan användas för att hitta litteratur kopplat till det testfall de kommer att jobba med. Tanken är att de på egen hand ska hitta litteratur som beskriver resultat och geometri om det behövs. Sedan formulerades frågor som studenterna skall besvara i sin rapport med hjälp av resultaten från deras simulering. Ur resultatet ska de ta fram grafer och bilder som underlag för att underbygga svaren på dessa frågor.

För att ge en bild av hur det förväntade resultatet ser ut så bifogas en bild. Denna bild gestaltar resultatet från simuleringar som genomförts på det testfall som uppgiften berör, "Laminärt visköst flöde över en platta". Detta för att ge en bild av hur resultatet bör se ut om man genomfört sina simuleringar korrekt.

Av de tre simuleringar som gjorts tas resultat från ett av de genomförda med Ducros-sensorn. Detta eftersom Ducros-sensorn gör att resultatet blir mer korrekt i jämförelse med Jameson-sensorn som fanns innan och att resultatet kan användas i uppgiften. Uppgiften finns i bilaga B.

Litteratur

[1] A. Jameson, W. Schmidt och E. Turkel, ”Numerical Solution of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time Stepping Schemes”, *AIAA*, årg. 14, s. 1–4, 1981. DOI: 10.2514/6.1981-1259.

[2] G. Degrez, C. H. Boccadoro och J. Wendt, ”The interaction of an oblique shock wave with a laminar boundary layer revisited. An experimental and numerical study”, *Journal of Fluid Mechanics*, årg. 177, s. 247–263, 1987. DOI: https://doi.org/10.1017/S0022112087000946.

[3] P. Petrie-Repar och P. Jacobs, ”A computational study of shock speeds in high-performance shock tubes”, *Shock Waves*, årg. 8, s. 79–91, 1998. DOI: https://doi.org/10.1007/s001930050101.

[4] F. Ducros, V. Ferrand, F. Nicoud m. fl., ”Large-Eddy Simulation of the Shock/Turbulence Interaction”, *Journal of Computational Physics*, årg. 152, nr 2, s. 517–549, 1999, ISSN: 0021-9991. DOI: https://doi.org/10.1006/jcph.1999.6238. URL: https://www.sciencedirect.com/science/article/pii/S0021999199962381.

[5] J. D. A. Jr., *Modern compressible flow with a historical perspective*, 3rd Edition. McGraw Hill, 2002.

[6] H. K. Versteeg och W. Malalasekera, *An introduction to fluid dynamics, The finite volume method*, 2nd Edition. Pearson Education, 2007, kap. 1, ISBN: 978-0-13-127498-3.

[7] E. a. John F. Wendt, *Computational fluid dynamics an introduction*, 3rd Edition. Springer-Verlag Berlin Heidelberg, 2009.

[8] F. M. White, *FLUID MECHANICS, SEVENTH EDITION*, 7th Edition. McGraw Hill, 2009, kap. 1, ISBN: 978-0-07-352934-9.

[9] S. S. Kanwar, G. Dubey, M. Singh och G. S. Khanday, ”CFD Analysis of Normal Shock using Shock Tube with Five Species”, *International Journal of Science and Research (IJSR)*, årg. 4, 2015.

[10] H. S. Kalsi, ”Numerical Modelling of Shock Wave Boundary Layer Interactions in Aero-engine Intakes at Incidence”, 2018. DOI: https://doi.org/10.1115/GT2018-75872.

[11] G. Singh, J. Sharma, R. Arora och I. Sandhu, ”A computational study of shock speeds in high-performance shock tubes”, *Materials Today: Proceedings*, årg. 28, s. 1872–1878, 2020. DOI: https://doi.org/10.1016/j.matpr.2020.05.294..

[12] E. B. I. NASA, *Blue Origin's Space Vehicle testing*, [Online; accessed 8 februari, 2023]. URL: https://quest.eb.com/search/132_1498516/1/132_1498516/cite.

[13] E. B. I. NASA, *Supersonic shock waves*, [Online; accessed 6 mars, 2023]. URL: https://quest.eb.com/search/132_3026931/1/132_3026931/cite.

A

Kod för beräkning av sensorer

A.1 Tidigare kod för hantering av stötar

```
if( self.__num._fcntp > 0. ):
        self.__sens_a[fmp_slice] = np.abs( np.add(np.subtract(P[FMP._kc
            [fmp_slice,0]], 2.*P[FMP._kc[fmp_slice,1]]), P[FMP._kc[
            fmp_slice,2]]))/\
        np.abs( np.add(np.add(P[FMP._kc[fmp_slice,0]],2.*P[FMP._kc[
            fmp_slice,1]]), P[FMP._kc[fmp_slice,2]]))
        self.__sens_b[fmp_slice] = np.abs( np.add(np.subtract(P[FMP._kc
            [fmp_slice,1]], 2.*P[FMP._kc[fmp_slice,2]]), P[FMP._kc[
            fmp_slice,3]]))/\
        np.abs( np.add(np.add(P[FMP._kc[fmp_slice,1]],2.*P[FMP._kc[
            fmp_slice,2]]), P[FMP._kc[fmp_slice,3]]))
        self.__cfilt[fmp_slice] self.__num._fcntp*np.maximum(self.
            __sens_a[fmp_slice],self.__sens_b[fmp_slice])*FMP._sl[
            fmp_slice]*(np.add(np.abs(self.__u_ave[fmp_slice]),self.
            __c_ave[fmp_slice]))
```

A.2 Ny kod för hantering av stötar

```
self.__cfilt[fmp_slice] = 0.

                if( self.__num._fcntp > 0. ):
                        self.__sens_a[fmp_slice] = np.abs( np.add(np.
                            subtract(P[FMP._kc[fmp_slice,0]], 2.*P[FMP.
                            _kc[fmp_slice,1]]), P[FMP._kc[fmp_slice
                            ,2]]))/\
                        np.abs( np.add(np.add(P[FMP._kc[fmp_slice
                            ,0]],2.*P[FMP._kc[fmp_slice,1]]), P[FMP._kc
                            [fmp_slice,2]]))
                        self.__sens_b[fmp_slice] = np.abs( np.add(np.
                            subtract(P[FMP._kc[fmp_slice,1]], 2.*P[FMP.
                            _kc[fmp_slice,2]]), P[FMP._kc[fmp_slice
                            ,3]]))/\
                        np.abs( np.add(np.add(P[FMP._kc[fmp_slice
                            ,1]],2.*P[FMP._kc[fmp_slice,2]]), P[FMP._kc
                            [fmp_slice,3]]))
                        self.__ducross1 = np.ones(self.__sens_a.size)
                        self.__ducross2 = np.ones(self.__sens_b.size)

                if( self.__num._fcntc > 0. ):
```

```
                self.__sens_a[fmp_slice] = np.abs( np.add(np.
                    subtract(RO[FMP._kc[fmp_slice,0]], 2.*RO[
                    FMP._kc[fmp_slice,1]]), RO[FMP._kc[
                    fmp_slice,2]]))/\
                np.abs( np.add(np.add(RO[FMP._kc[fmp_slice,0]],
                     2.*RO[FMP._kc[fmp_slice,1]]), RO[FMP._kc[
                    fmp_slice,2]]))
                self.__sens_b[fmp_slice] = np.abs( np.add(np.
                    subtract(RO[FMP._kc[fmp_slice,1]], 2.*RO[
                    FMP._kc[fmp_slice,2]]), RO[FMP._kc[
                    fmp_slice,3]]))/\
                np.abs( np.add(np.add(RO[FMP._kc[fmp_slice,1]],
                     2.*RO[FMP._kc[fmp_slice,2]]), RO[FMP._kc[
                    fmp_slice,3]]))
                self.__cfilt[fmp_slice] = np.maximum(self.
                    __cfilt[fmp_slice],self.__num._fcntc*np.
                    maximum(self.__sens_a[fmp_slice],self.
                    __sens_b[fmp_slice])*FMP._sl[fmp_slice]*np.
                    abs(self.__u_ave[fmp_slice]))


        if( FMP.face_type != FMPTypes.BOUNDARY and g3dworld.
            shock_damping_mode.value > ShockDampingMode.JAMESON
            .value ):
                if( g3dworld.shock_damping_mode is
                    ShockDampingMode.DUCROSS_FACE_GRADIENTS ):

                    # EVALUATE DUCROSS SENSOR WEIGHT FACTOR
                        USING FACE-BASED VELOCITY
                        GRADIENTS

                    self.__dudx[fmp_slice], self.__dudy[
                        fmp_slice], self.__dudz[fmp_slice]
                        = self.__CalculateFaceGradient(U,
                        FMP,fmp_slice)
                    self.__dvdx[fmp_slice], self.__dvdy[
                        fmp_slice], self.__dvdz[fmp_slice]
                        = self.__CalculateFaceGradient(V,
                        FMP,fmp_slice)
                    self.__dwdx[fmp_slice], self.__dwdy[
                        fmp_slice], self.__dwdz[fmp_slice]
                        = self.__CalculateFaceGradient(W,
                        FMP,fmp_slice)
                    div_u = np.add(np.add(self.__dudx,self.
                        __dvdy),self.__dwdz)
                    curl_u = np.add(np.add(np.subtract(self
                        .__dwdy, self.__dvdz),np.subtract(
                        self.__dudz, self.__dwdx)), np.
                        subtract(self.__dvdx, self.__dudy))
                    self.__ducross1 = np.divide(np.multiply
                        (div_u,div_u),np.add(np.multiply(
                        div_u,div_u),np.add(np.multiply(
                        curl_u,curl_u),np.multiply(1e-30,
                        np.ones(self.__dudx.size)))))

                    self.__ducross2 = np.divide(np.multiply
```

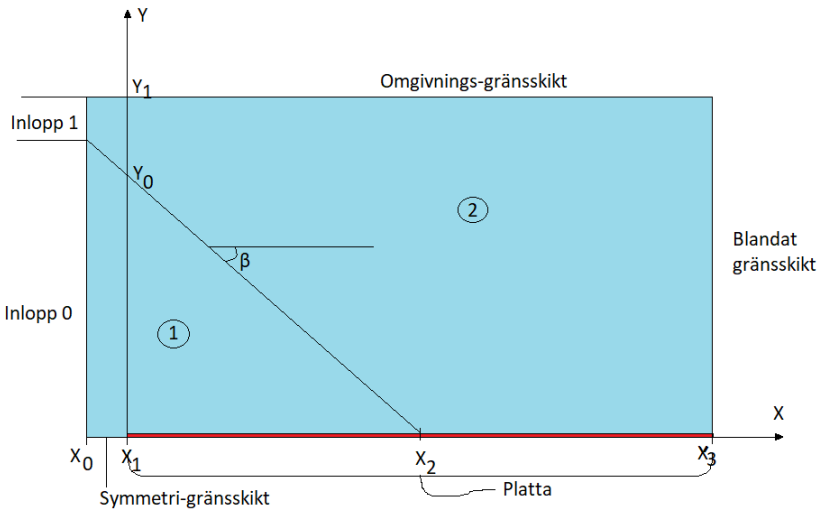
```
            (div_u,div_u),np.add(np.multiply(
            div_u,div_u),np.add(np.multiply(
            curl_u,curl_u),np.multiply(1e-30,
            np.ones(self.__dudx.size)))))

    elif( g3dworld.shock_damping_mode is
        ShockDampingMode.DUCROSS_CELL_GRADIENTS ):

            # EVALUATE DUCROSS SENSOR WEIGHT FACTOR
                USING CELL-BASED VELOCITY
            GRADIENTS

            test = 1.
            DUDX, DUDY, DUDZ, DVDX, DVDY, DVDZ,
                DWDX, DWDY, DWDZ = self.__domain.
                GetVGradArrays()
            div_u1 = np.add(np.add(DUDX[FMP._kc[
                fmp_slice, 2]], DVDY[FMP._kc[
                fmp_slice, 2]]), DWDZ[FMP._kc[
                fmp_slice, 2]])

            curl_u1 = np.add(np.add(np.subtract(
                DWDY[FMP._kc[fmp_slice, 2]], DVDZ[
                FMP._kc[fmp_slice, 2]]), np.
                subtract(DUDZ[FMP._kc[fmp_slice,
                2]], DWDX[FMP._kc[fmp_slice, 2]])),
                 np.subtract(DVDX[FMP._kc[fmp_slice
                , 2]], DUDY[FMP._kc[fmp_slice, 2]])
                )

            div_u2 = np.add(np.add(DUDX[FMP._kc[
                fmp_slice, 3]], DVDY[FMP._kc[
                fmp_slice, 3]]), DWDZ[FMP._kc[
                fmp_slice, 3]])
            curl_u2 = np.add(np.add(np.subtract(
                DWDY[FMP._kc[fmp_slice, 3]], DVDZ[
                FMP._kc[fmp_slice, 3]]), np.
                subtract(DUDZ[FMP._kc[fmp_slice,
                3]], DWDX[FMP._kc[fmp_slice, 3]])),
                 np.subtract(DVDX[FMP._kc[fmp_slice
                , 3]], DUDY[FMP._kc[fmp_slice, 3]])
                )

            self.__ducross1 = np.divide(np.multiply
                (div_u1,div_u1),np.add(np.multiply(
                div_u1,div_u1),np.add(np.multiply(
                curl_u1,curl_u1),np.multiply(1e-30,
                 np.ones(DUDX[FMP._kc[fmp_slice,
                2]].size)))))

            self.__ducross2 = np.divide(np.multiply
                (div_u2,div_u2),np.add(np.multiply(
                div_u2,div_u2),np.add(np.multiply(
                curl_u2,curl_u2),np.multiply(1e-30,
                 np.ones(DUDX[FMP._kc[fmp_slice,
                3]].size)))))
```

```
# COMPUTE DUCROSS OR JAMESON SENSOR
if( self.__num._fcntp > 0. ):
        self.__cfilt[fmp_slice] = self.__num._fcntp*np.
            maximum(np.multiply(self.__sens_a[fmp_slice
            ], self.__ducross1[fmp_slice]),np.multiply(
            self.__sens_b[fmp_slice], self.__ducross2[
            fmp_slice]))*FMP._sl[fmp_slice]*(np.add(np.
            abs(self.__u_ave[fmp_slice]),self.__c_ave[
            fmp_slice]))
```

B

Testcase

Inviscid compressible flow in two space dimensions. A geometry according to the picture below is defined and the problem consists of computing numerically the 2D steady-state flow including compression regions and shocks. The flow domain consists of a plate and an upper free boundary, as well as a symmetry boundary in front of the plate. A supersonic flow is applied at the inlet.



Geometry		
Inlet [m]	$y_0 = 1.2 * tan(\beta) = 0.71534356847$	
	$y_1 = 1$	
Plate [m]	$x_0 = -0.2$	$x_2 = 1$
	$x_1 = 0$	$x_3 = 2$

Inlet		Outlet
Inlet 0	Inlet 1	

$M_{\infty} = 2.15$	$M_1 = 2.01$	$M_2 = 2.01$
$T_{\infty} = 288.00\ K$	$T_1 = 306.87\ K$	$T_2 = 306.87\ K$
$P_{\infty} = 0.01\ bar$	$P_1 = 1.26\ bar$	$P_2 = 1.26\ bar$
$\rho_{\infty} = 1.23\ \frac{kg}{m^3}$	$\rho_1 = 1.44\ \frac{kg}{m^3}$	$\rho_2 = 1.44\ \frac{kg}{m^3}$
$a_{\infty} = 340.17\ \frac{m}{s}$	$a_1 = 351.14\ \frac{m}{s}$	$a_2 = 351.14\ \frac{m}{s}$

Expected outcome:

You are expected to observe a second oblique shock wave reflected from the first oblique shock wave as well as expansion fans emanating from where the first oblique shock wave makes contact with the plate.

Questions:
Why are there expansion fans in the flow above the second oblique shock?
What happens in the boundary layer that causes the shocks to appear?
Does the shock make the boundary layer or does the boundary layer cause the shock?
Based on the results of the simulation, explain what happens in the flow and why?

Suggested search topic:
Oblique shock wave with a laminar boundary layer.

INSTITUTIONEN FÖR MEKANIK OCH MARITIMA VETENSKAPER
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2023
www.chalmers.se

CHALMERS