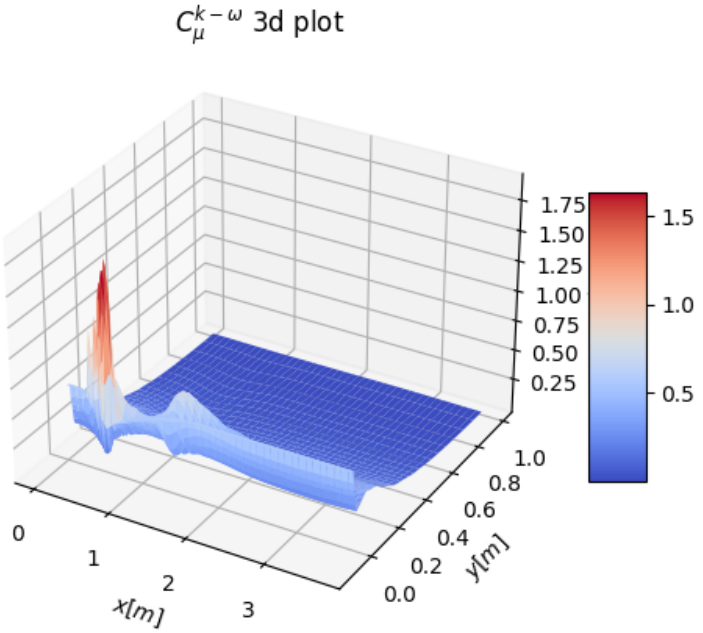


Utveckling av turbulensmodeller med hjälp av maskininlärning i Python

Kandidatarbete vid institutionen för Mekanik och Maritima vetenskaper (M2)

Batlouni, Fadi
Elm Jonsson, Benjamin
Fjeldså, Ole
Persson, Niclas
Ånestrand, Leo

Institutionen för Mekanik och Maritima vetenskaper (M2)

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

Kandidatarbete 2023

Utveckling av turbulensmodeller med hjälp av maskininlärning i Python

Kandidatarbete vid institutionen för Mekanik och Maritima vetenskaper (M2)

CHALMERS

Institutionen för Mekanik och Maritima vetenskaper (M2)
Avdelningen för strömningsmekanik
Chalmers Tekniska Högskola
Göteborg, Sverige 2023

Utveckling av turbulensmodeller med hjälp av maskininlärning i Python
Kandidatarbete vid institutionen för Mekanik och Maritima vetenskaper (M2)

Batlouni, Fadi
Elm Jonsson, Benjamin
Fjeldså, Ole
Persson, Niclas
Ånestrand, Leo

Handledare: Davidsson, Lars, Institutionen för Mekanik och Maritima vetenskaper
Examinator: Andersson, Niklas, Instituionen för Mekanik och Maritima vetenskaper

Kandidatarbete MMSX21 2023
Institutionen för Mekanik och Maritima vetenskaper (M2)
Avdelningen för strömningsmekanik
Chalmers Tekniska Högskola
SE-412 96 Göteborg
Sverige
Telefon +46 31 772 1000

Omslagsbild: Bild som visar $C_{\mu}^{k-\omega}$, baserat på DNS data i domänen som ges av small wavefallet.

Illustrationerna i arbetet är författarnas egna om inget annat anges

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Utveckling av turbulensmodeller med hjälp av maskininlärning i Python
Kandidatarbete vid institutionen för Mekanik och Maritima vetenskaper (M2)

Batlouni, Fadi
Elm Jonsson, Benjamin
Fjeldså, Ole
Persson, Niclas
Årnestrand, Leo

Institutionen för Mekanik och Maritima vetenskaper (M2)
Chalmers University of Technology

Abstract

Turbulence modelling is a central component in the field of *computational fluid dynamics* (CFD), which aims to accurately simulate turbulent flows. To do this, solving the Navier-Stokes equations numerically is necessary. However, due to the fact that turbulence is rather chaotic, *direct numerical simulation* (DNS) is computationally expensive. The idea of using *Reynold-Averaged Navier Stokes* (RANS) and turbulence models such as the $k-\omega$ model is to reduce the computational costs by simplifying the equations, at the cost of losing accuracy.

The main aim of the project is to explore if some specific parameters used in these simplified equations, usually assumed to be constant, can be optimized with the use of *machine learning* (ML), and how the improved models fare against previous models. The ML methods used are *support vector regression* (SVR), *k nearest neighbors* regression (kNN), and *neural networks*. The project found that optimizing C_μ with the $k-\omega$ model is inessential, since the model badly predicts k even though the fraction of k and ω can be correct making the optimization difficult. Nonetheless, optimizing other parameters still proves to be rewarding. The use of machine learning to improve turbulence models is considered promising and should be explored further.

Keywords: Maskininlärning, turbulensmodellering, CFD, DNS, SVR, RANS, Python, strömningsmekanik

Tack

Vi tackar ödmjukast för all hjälp vi har fått av vår handledare, Lars, utan dig hade detta projekt ej kunnat genomföras. Den kunskap inom flertalet områden som projektgruppen ackumelerat under projektets gång är avsevärd, vilket vi är tacksamma för att ha fått möjligheten att tillgå.

Dessutom vill vi tacka examinatorn, Niklas, som tagit sig tiden att ge feedback på arbetet.

Till sist vill projektgruppen tacka sina respektive familjer och vänner för all uppmuntran och stöd som givits under projektets gång.

Göteborg, Juni 2023

Akronymer

Nedan listas de akronymer som är genomgående i arbetet.

CFD	Computational Fluid Dynamics
DNS	Direkt numerisk Simulering
kNN	k-Nearest Neightbors
ML	Machine Learning
MAPE	Mean Average Percentage Error
MAE	Mean Average Error
MSE	Mean Squared Error
NEVM	Non-linear Eddy Viscosity Model
RANS	Reynolds-Averaged Navier Stokes
RMS	Root Mean Square
RMSE	Root Mean Squared Error
SVR	Support Vector Regression
SVM	Support Vector Machine
SSR	Sum of Squared Residuals
SGD	Stochastic Gradient Descent

Nomenklatur

Nedan följer den aktuella nomenklaturen för rapporten, uppdelad i Strömningsmekanik och Maskinilärning. Detta innefattar index, parametrar, variabler, och övrig notation som kommer användas genomgående i rapporten.

Strömningsmekanik

Index

i,j	Index
c	Cell-center

Parametrar

$C_\mu^{k-\varepsilon} = 0.09$	Konstant i $k-\varepsilon$ modellen
$C_\mu^{k-\omega} = 1$	Konstant i $k-\omega$ modellen
c_1 = -0.05	Konstant i den ickelinjära *Eddy-viscousity* modellen
$c_2 = 0.11$	Konstant i den ickelinjära *Eddy-viscousity* modellen
$c_3 = 0.21$	Konstant i den ickelinjära *Eddy-viscousity* modellen
ρ	Densitet

Variabler

C_μ	Eftersökt variabel
c_1	Eftersökt variabel ($c_1 = -0.05$ antas annars)
c_2	Eftersökt variabel ($c_2 = 0.11$ antas annars)
c_3	Eftersökt variabel ($c_3 = 0.21$ antas annars)
k	Turbulent kinetisk energi
τ	Turbulent tidsskala
ε	Dissipation av turbulent kinetisk energi

ν	Kinematisk viskositet
ν_t	Kinematisk turblent viskositet
u, v, w	Momentanhastigheten i x-led, y-led respektive z-led
$\overline{u}$, $\overline{v}$, $\overline{w}$	Tids-genomsnittet av hastigheten i x-led, y-led respektive z-led
$\overline{u'u'}$, $\overline{v'v'}$, $\overline{w'w'}$	Turbulent Reynold spänningar
$\frac{\partial \overline{u}}{\partial x}$, $\frac{\partial \overline{v}}{\partial x}$	Hastighetsutveckling i x-led
$\frac{\partial \overline{u}}{\partial y}$, $\frac{\partial \overline{v}}{\partial y}$	Hastighetsutveckling i y-led
p	Tryck
X_c, Y_c	x och y-koordinat för cell-center
y^+	Dimensionslöst avstånd från vägg i kanalströmning

Tensorer

a_{ij}	Anisotropiska Reynolds spänningstensor
δ_{ij}	Kroneckerdelta
S_{ij}	Strain - rate tensor

Maskinlärning

Index

i,j	Index

Parametrar

C	Hyperparameter för SVR
ξ	*Slack*
ε	Hyperplansmarginal
k	Antal datapunkter (k Nearest Neighbor)
k	Dimension (k-d Tree)
n	Antal datapunkter/Antal noder
p	Allmänt namn för parameter
α	Steglängd

Variabler

y_{real}	Faktiska y-värden
y_{pred}	Förutspådda y-värden
θ	Bias
$x_1, x_2, \ldots, x_n$	Indata
$w_1, \ldots, w_4$	Vikter
O	Utdata
$s_1, s_2, \ldots, s_n$	Värden på noder

Funktioner och övrig notation

Σ	Summa
$\mathcal{O}$	Komplexitet och gränsbeteende
g	Aktiveringsfunktion
std	standardavvikelse
$mean$	Medelvärdet

Innehåll

Figurer

Tabeller

Listings

1
Introduktion

Turbulensmodellering är en viktig del av flera tekniska tillämpningar, såsom flygindustri, rymdteknik, miljö- och energiteknik [1]. Ett exempel på turbulensmodellering är simuleringen av luftströmning runt flygplanets vingar för att förutsäga dess aerodynamiska egenskaper. Andra tillämpningar inkluderar simuleringar av strömmar i rör, i kompressorer och i förbränningsprocesser.

Traditionellt sett har turbulensmodellering baserats på ett flertal teoretiska antagande om strömningen, vilket innebär att turbulensbeskrivning görs på en makroskopisk skala, baserat på empiriska data. Denna metod har dock vissa begränsningar, eftersom turbulens är mycket komplex och svår att förutsäga. Dessutom är traditionella modeller ofta specifika för en viss typ av strömning och kan inte generaliseras till andra situationer.

Maskininlärning (ML) har funnits länge men utveckling och användning har drastiskt ökat de senaste åren. Vanliga applikationer för ML är klassificering och regression. Exempel på ett klassifieringsproblem är när datan består av foton, där ML används för att känna igen trafikskyltar eller trafikljus [2]. Ett annat exempel är igenkänning av hjärtproblem där datan som används är Elektrokardiografi [3]. När datan rör sig på en skala snarare än att de har entydiga klassifieringar har man istället ett regressionsproblem. Prediktion av bromsavståndet för bilar givet vilket hastighet de kör i är ett exempel på ett sådant problem.

Med utvecklingen av ML finns möjligheten att tillämpa en mer flexibel och generell metod för turbulensmodellering. ML är en metod som kan hitta komplexa mönster i data genom att träna regressionsmodeller, neurala nätverk eller andra algoritmer. Detta innebär att ML kan hjälpa till att utveckla en modell som presterar bättre än traditionella modeller och som kan anpassas till olika typer av strömningar.

1.1 Bakgrund

Inom strömningslära är navier-stokes ekvation grundläggande. Ekvationen har ännu ej lösts anylytiskt utan vill man lösa den använder man sig idag av direkt numerisk simulering (DNS). Problemet med DNS är att det kan ta flera månader att lösa med dagens beräkningskraft. Av den anledningen använder man ofta enklare turbulensmodeller. Turbulensmodellerna erbjuder snabbare beräkningstid på bekostnad av lägre träffsäkerhet och begränsningar på vilka flöden som kan modellen kan appliceras på. Dessa modeller, exempelvis k-ω modellen, använder sig av konstanter framtagna från empiriska studier.

Som i många andra fall med strömningsmekanik så utgås det från Navier Stokes ekvation. Fokuset kommer ligga på Time Averaged Navier Stokes ekvationerna, även kallad Reynolds Averaged Navier Stokes eller RANS-ekvationerna i två dimensioner och Raynolds-spänningen för turbulenta flöden. Mer specifikt tittas det närmare på en parameter, C_μ. I nuvarande turbulensmodeller brukar C_μ approximeras som konstant, vilket historiskt sett fungerat ganska bra. Vårt projekt ska utreda för huruvida ett varierande C_μ kan resultera i mer träffsäkra modeller.

Det finns dock inget som antyder att dessa konstanter faktiskt bör vara konstanter. Det finns därmed möjlighet att utforska ifall dessa modeller skulle prestera bättre om varje konstant istället representeras som en funktion, med beroende på en eller flera egenskaper i flödet. Med hjälp av ML kan nya modeller för dessa funktioner tas fram.

1.2 Syfte

Syftet är att med hjälp av ML förbättra nuvarande turbelensmodeller för att få en mer korrekt beskrivning av flöde.

1.3 Problem

Med hjälp av *Support vector regression* (SVR) och korrekt DNS-data ska en modell där konstanter byts ut mot variabler skapas för att förutspå $C_\mu^{k-\omega}$. Denna modell ska i sin tur testas på annan DNS data och förutså C_μ. Sedan jämför felet mellan den förutspådda resultatet och standardmodellen där $C_\mu = 1$ med det korrekta värdet (DNS) för att se om den nya modellen presterar bättre. I mån av tid ska även andra, alternativa ML-metoder testas. Närmre bestämt k-d Tree, k-Nearest Neighbors (kNN) och Neurala nätverk.

Utöver optimering av C_μ så ska även konstanterna $c_1 - c_3$ från Non-linear Eddy-viscosity modellen optimeras.

1.4 Avgränsningar och antaganden

Detta avsnitt syftar till att ta upp begränsningarna i denna kandidatuppsats och ge en transparent och kritisk bedömning av studiens begränsningar. Trots betydande ansträngningar för att genomföra en noggrann undersökning och dra meningsfulla slutsatser är det viktigt att erkänna de inneboende begränsningar som kan påverka generaliserbarheten och validiteten hos resultaten. Dessa begränsningar omfattar olika aspekter, inklusive val av data, metod, teoretiska antaganden, tids- och resursbegränsningar och studiens omfattning. Genom att öppet diskutera dessa begränsningar syftar projektgruppen till att öka förståelsen och tolkningen av resultaten samtidigt som vi uppmuntrar framtida studier att hantera dessa begränsningar och bygga vidare på detta arbete.

Arbetet är begränsat till tidsperioden från januari 2023 till maj 2023, där varje medlem i projektet har tilldelats 20 timmar per vecka för att ägna åt arbetet.

En central komponent i detta arbete är den data som kommer användas för olika fall av strömning, vilket utgör grunden för hela studien. Denna data hämtas från externa källor och genereras inte internt av gruppen på grund av tids- och kunskapsbegränsningar. Resultatet är starkt beroende av denna data och det kan inte garanteras att samma resultat skulle erhållas för andra fall, även om syftet är att uppnå så generella resultat som möjligt.

För att möjliggöra beräkningarna har detta arbete gjort flera antaganden om strömningen. Ett grundläggande antagande är att massa, energi och moment antas vara bevarade i strömningen. Dessutom antas Boussinesq-antagandet gälla för fallen *small wave* och *large wave* där beroendet av densiteten. Antagandet är att fluiden är inkomressibel, vilket innebär att variation av massa och densitet ignoreras.

De metoder som kommer användas i denna undersökning begränsas till SVR, regression med neurala nätverk, och kNN.

2
Teori

I följande del av rapporten presenteras relevant teori bakom strömmingsmekanik, samt ML-metoderna SVR, kNN, och Neuralt Nätverk. Denna teori ligger till grund för projektet samt projektgruppens val av metod.

2.1 Strömningsmekanik

Turbulensmodellerna som projektgruppen kommer att utgå ifrån är $k-\omega$ modellen och en icke-linjär eddy-viscosity-modell. De bygger på RANS-ekvationen som uttrycks i ekvation 2.1. De två sista termerna är okända Reynold-spänningar som måste modelleras.

$$\underbrace{\overline{u}\frac{\partial \overline{u}}{\partial x}+\overline{v}\frac{\partial \overline{u}}{\partial y}}_{\text{Derivata av hastighetsfält}} = \underbrace{-\frac{1}{\rho}\frac{\partial \overline{p}}{\partial x}}_{\text{Tryckgradient}} + \underbrace{\nu\left(\frac{\partial^2 \overline{u}}{\partial x^2}+\frac{\partial^2 \overline{u}}{\partial y^2}\right)}_{\text{Viskös}} \underbrace{-\frac{\partial \overline{u'u'}}{\partial x}-\frac{\partial \overline{u'v'}}{\partial y}}_{\text{Reynold-spänningar}} \tag{2.1}$$

Genom att använda sig av Boussinesq-antagadet där man introducerar turbulent viskositet, ν_t fås ekvation 2.3 och beroendet av densiteten försvinner. [4, p.129]

$$\overline{v_i'v_j'} = \nu_t\left(\frac{\partial \overline{u_i}}{\partial x_j}+\frac{\partial \overline{u_j}}{\partial x_i}\right)-\frac{2}{3}k\delta_{ij} \tag{2.2}$$

$$\text{där } \delta_{ij} = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases} \tag{2.3}$$

I $k-\omega$ modellen modelleras spänningarna enligt boussinesq-antagandet i ekvation 2.4.

$$\overline{u'v'} = -\nu_t\left(\frac{\partial \overline{u}}{\partial y}+\frac{\partial \overline{v}}{\partial x}\right) \tag{2.4}$$

2.1.1 $k-\omega$ modellen

I $k-\omega$ modellen, som föreslogs av David C. Wilcox(2006), löses standard-ekvationen för k men använder en längdbestämmande ekvation, ω. Variabel ω kallas ofta för specifik spridning från sin definition i ekvation 2.5. [5]

$$\omega \propto \frac{\varepsilon}{k} \tag{2.5}$$

Standardekvationen för k är definerad enligt ekvation 2.6 där k är summan av de normala reynold-spänningarna.

$$k = \frac{1}{2}\sum_{i=1}^{3}\overline{v_i'^2} = \frac{1}{2}(\overline{u'^2} + \overline{v'^2} + \overline{w'^2}) \equiv \frac{1}{2}\overline{v_i' v_i'} \tag{2.6}$$

Uttrycken i ekvation 2.7 fås genom omskrivning av $k-\varepsilon$ modellen [4, kap.16, p. 178].

$$\nu_t = \frac{k}{\omega}, \quad \omega = \frac{\varepsilon}{C_\mu^{k-\varepsilon} k} \tag{2.7}$$

$$\nu_t \equiv C_\mu^{k-\omega}\frac{k}{\omega} \tag{2.8}$$

Variabeln C_μ kan sedan räknas ut med hjälp av ekvation 2.4 och ekvation 2.8 och skrivas om till ekvation 2.9.

$$C_\mu^{k-\omega} = -\overline{u'v'}\left(\frac{\partial \overline{u}}{\partial y} + \frac{\partial \overline{v}}{\partial x}\right)^{-1}\frac{\omega}{k\cdot} \tag{2.9}$$

2.1.2 Ickelinjär Eddy-viscosity Modell

Den icke-linjär Eddy-viscosity-modellen (NEVM, från engelskan) formuleras från Boussinesq-antagandet och den normaliserade anisotropiska Reynolds spänningstensor, a_{ij}.[4] Med den normaliserade anisotropiska Reynolds spänningstensorn innebär att den tensorn har olika fysikaliska egenskaper i olika riktningar och användadet av tensorn ger möjlighet till att reducera det tredimensionella flödet till två konstanta variabler som beskriver flödet tvådimensionellt. Tensorn beskrivs enligt ekvation 2.10.

$$a_{ij} \equiv \frac{\overline{v_i' v_j'}}{k} - \frac{2}{3}\delta_{ij} \tag{2.10}$$

För fullt utvecklad kanalströmming, $\overline{v}_2 = \overline{v}_3 = \partial/\partial x_1 = \partial/\partial x_3 \equiv 0$, fås då ekvationerna 2.11-2.13. Det vill säga då flödet i exempelvis ett rör är stationärt.

$$a_{11} = \frac{1}{12}\tau^2 \left(\frac{\partial \overline{v_1}}{\partial x_2}\right)^2 (c_1 + 6c_2 + c_3), \tag{2.11}$$

$$a_{22} = \frac{1}{12}\tau^2 \left(\frac{\partial \overline{v_1}}{\partial x_2}\right)^2 (c_1 - 6c_2 + c_3), \tag{2.12}$$

$$a_{33} = -\frac{1}{6}\tau^2 \left(\frac{\partial \overline{v_1}}{\partial x_2}\right)^2 (c_1 + c_3), \tag{2.13}$$

Den turbulenta tidsskalan, τ, skrivs om med hjälp av enhetsanalys och används enligt definition i ekvation 2.14.

$$\tau = \frac{k}{\varepsilon} \tag{2.14}$$

Ekvation 2.11 - 2.14 löses vanligt med vedertagna konstanter från [6]. Dessa lyder enligt följande.

$$c_1 = -0,05\frac{f_q}{f_\mu} = -0,05 \tag{2.15}$$

$$c_2 = 0,11\frac{f_q}{f_\mu} = 0,11 \tag{2.16}$$

$$c_3 = 0,21\frac{f_q \tilde{S}}{f_\mu(\tilde{S} + \tilde{\Omega})/2} = 0,21 \tag{2.17}$$

Det vill säga, då konstanterna ovan används sätts $\frac{f_q}{f_\mu} = \frac{f_q \tilde{S}}{f_\mu(\tilde{S}+\tilde{\Omega})/2} = 1$.

För att se till att systemet inte ska få singulariteter så används sambandet i ekvation 2.18 och ekvation 2.19.

$$a_{ii} \equiv a_{11} + a_{22} + a_{33} = 0 \tag{2.18}$$

$$c_0 \equiv c_1 + c_3 \tag{2.19}$$

Ekvationerna 2.10 - 2.13 kan skrivas om till ekvationerna 2.20 - 2.21 där då c_0 kan bestämmas, som visas i ekvation 2.22

$$a_{11} = \frac{1}{12}\tau^2 \left(\frac{\partial \overline{v}_1}{\partial x_2}\right)^2 (c_0 + 6c_2), \tag{2.20}$$

$$a_{33} = -\frac{1}{6}\tau^2 \left(\frac{\partial \overline{v}_1}{\partial x_2}\right)^2 c_0, \tag{2.21}$$

Ekvationerna 2.20 och 2.21 kan då skrivas om till att lösa ut c_2 i ekvation 2.23, där a_{ij} kan bestämmas med hjälp av ekvation 2.10.

$$c_0 = -\frac{6a_{33}}{\tau^2(\frac{\partial \overline{v}_1}{\partial x_2})^2}. \tag{2.22}$$

$$c_2 = \frac{2a_{11}}{\tau^2(\frac{\partial \overline{v}_1}{\partial x_2})^2} - \frac{c_0}{6} = \frac{2a_{11} + a_{33}}{\tau^2(\frac{\partial \overline{v}_1}{\partial x_2})^2} \tag{2.23}$$

Genom att använda c_0 och c_2 kan ekvationerna 2.24 - 2.26 ställas upp för att beräkna Reynolds spänning och jämföras med de erhållna från DNS.

$$\overline{u'u'} = \left(\frac{1}{12}\tau^2(\frac{\partial \overline{v}_1}{\partial x_2})^2(c_0 + 6c_2) + \frac{2}{3}\right)k \tag{2.24}$$

$$\overline{v'v'} = \left(\frac{1}{12}\tau^2(\frac{\partial \overline{v}_1}{\partial x_2})^2(c_0 - 6c_2) + \frac{2}{3}\right)k \tag{2.25}$$

$$\overline{w'w'} = \left(-\frac{1}{6}\tau^2(\frac{\partial \overline{v}_1}{\partial x_2})^2 c_0 + \frac{2}{3}\right)k \tag{2.26}$$

2.2 Maskininlärning

I detta projekt implementeras ML-metoder utvecklade för lösning av regressionsproblem. Metoderna SVR, kNN-regression, och neurala nätverk används, och i detta kapitel beskrivs teorin bakom dem. Förklaringarna är ofta inte fullständiga, så läsare hänvisas till citerad litteratur för djupare förståelse.

2.2.1 Skalering

Innan ML borde datan skaleras. Skaleringens syfte är att transformera värderna i inparametrar så de hamnar i en ”liknade” skala.

Huvudsyftet med skalering av indata är att säkra att alla inparametrar bidrar likvärdigt till resultatet även om parametrarna rör sig på olika storleksskalor. Detta är särskilt viktigt för algoritmer som förlitar sig på avstånd innom datan för att förustspå målvärden, som SVR och kNN. Skalering av indata kan också skynda på konvergeringen för algoritmer som använder gradientnedstigning, exempelvis neurala nätvärk.

Några vanliga Skaleringsfunktioner är:

- **Standardisering**
  Standardisering transformerar datan sådan att genomsnittet $x' = \frac{x-\mu}{\sigma}$, där x är den orginala inputdatan, x' är den skalerade inputdatan, μ är genomsnitt av x, och σ är standardavvikelsen av x. Denna metod transformerar datan så att den får genomsnittet 0 och standardavvikelse 1.

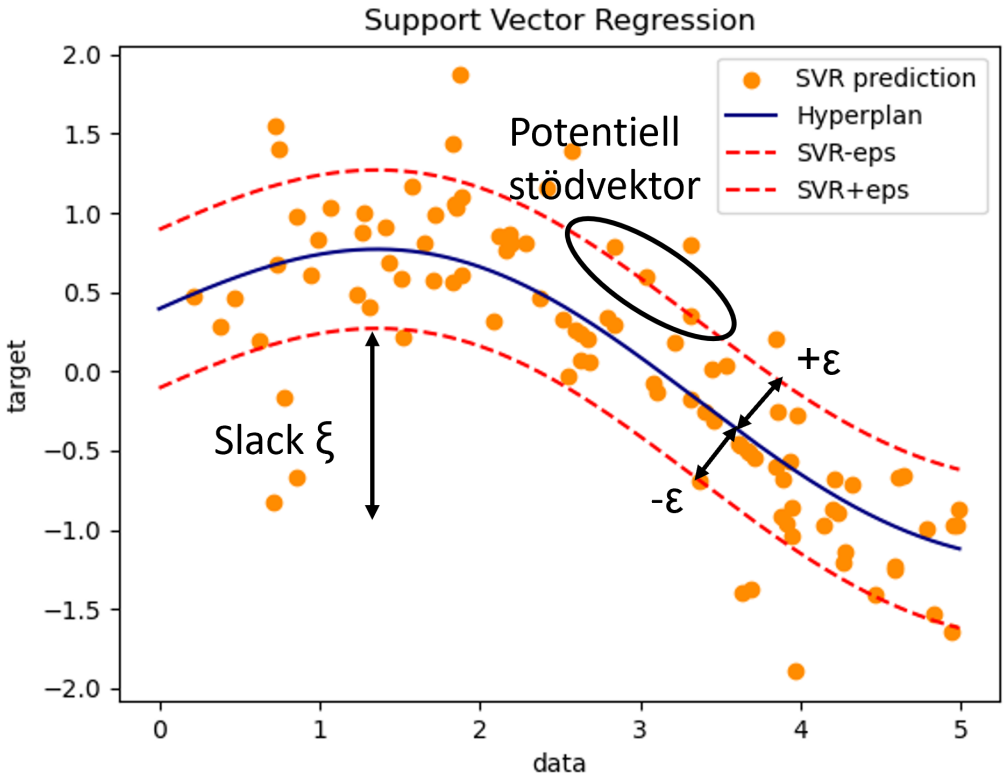
- **Min-Max-skalering**
  Skalerar så datan hamnar innom något intervall, typiskt $[0, 1]$
  $x' = \frac{x-x_{min}}{x_{max}-x_{min}}$

- **Logaritmisk skalering**
  $x' = log(x)$

Eftersom genomsnitt och standardavvikelser kan variera mellan dataset är det god praksis att spara skaleringsparametrar tillsammans med den färdigtränade modellen så ny data kan skaleras på samma vis som modellens träningsdata.

2.2.2 Support vector regression

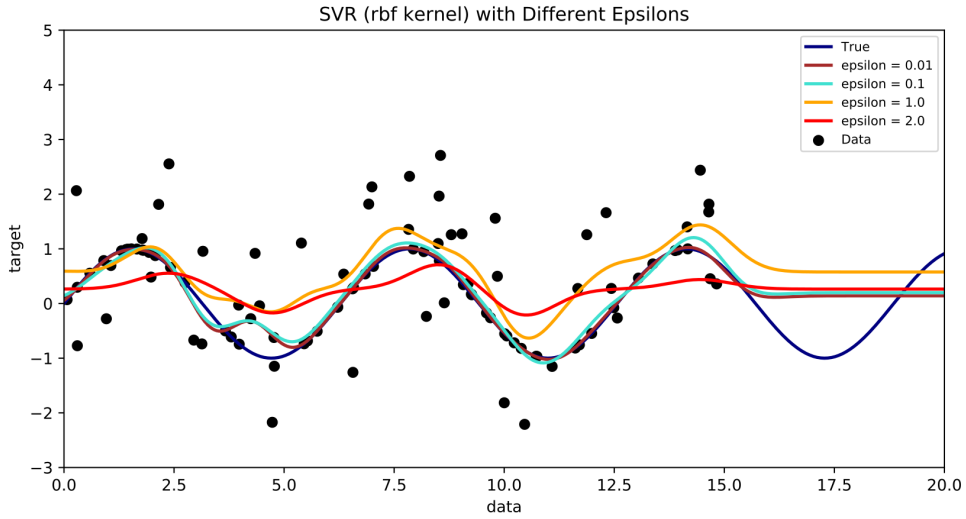
SVR är en ML-algoritm som används för att prediktera kontinuerliga numeriska värden. Precis som stödvektormaskiner (SVM) är SVR en typ av övervakad ML, vilket betyder att den tränas på märkta exempel där varje exempel har en etikett som representerar det faktiska värdet[7]. SVR är lik en vanlig linjär regression, med skillnaden att SVR kan modellera icke-linjära förhållanden mellan olika variabler. SVR försöker hitta en funktion som bäst passar till träningsdatan genom att minimera en förlustfunktion som mäter avståndet mellan den faktiska och förutsagda utgången. Denna funktion är konstruerad med hjälp av stödvektorer, som är de exempel som ligger närmast beslutningsgränsen eller hyperplanet. Algoritmen har två viktiga hyperparametrar, toleransen ε och kostnaden C, där parametern C är en skalningsfaktor för slack, ξ som i sin tur är avvikelsen från toleransen.



Figur 2.1: Hyperplan, ε - marginal, ξ - slack och datapunkter.

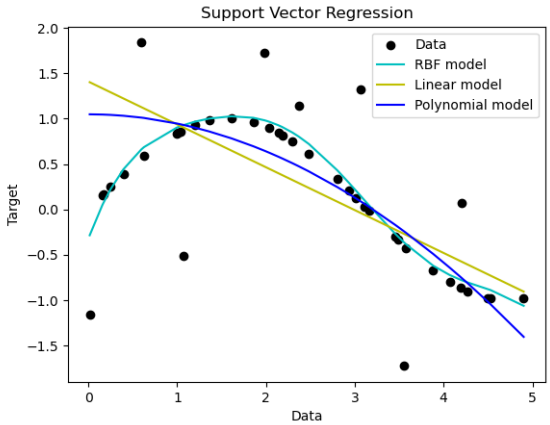
I figur 2.1 fungerar Epsilon, ε - parametern som en marginal runt hyperplanet som avgör hur nära förutspådd data *bör* vara träningsdatan. Den fungerar som en felmarginal. En hög epsilon-parameter innebär att modellen accepterar en högre felmarginal, medan en låg ε-parameter innebär att modellen kommer att försöka minimera felmarginalen. I figur 2.2 illusteras hur ε påverkar funktionen. Där ses att när C är låg så ignoreras många av punkterna, vilket gör hyperplanet mer distinkt medan ett högre C ger ett mjukare hyperplan som anpassar sig efter fler punkter.

Slack, ξ - justeras av "C", där ett högre C leder till ett mindre ξ. C - parametern kontrollerar balansen mellan att ha en mjukare beslutningsgräns och att ha färre avvikande extrempunkter. Ett högt C-värde innebär att modellen kommer att försöka minimera felaktigheter på bekostnad av att tillåta fler avvikande punkter, medan ett lågt C-värde innebär att modellen kommer att fokusera mer på att hitta en bredare beslutningsgräns och därmed inkludera fler avvikande punkter. Ett högre C-värde leder alltså till bättre passning på träningsdatan, vilket ofta ger ett mindre fel. Problemet är att det kan leda till överpassning vilket ger sämre prestering på nya datamängder. Ett lågt C-värde innebär mindre känslighet för för avvikande punkter men också mindre flexibilitet.



Figur 2.2: SVR med olika epsilon från [8] CC BY-SA 4.0

SVR har fyra inbyggda kärnfunktioner. Dessa funktioner transformerar indatan till högre dimensioner där den försöker hitta ett lämpligt hyperplan. Valet av kärnfunktion påverkar i hög grad hyperplanet 2.3.



Figur 2.3: Hyperplan beroende på vald kärnfunktion

De fyra kärnfunktioner är följande:

- **Linjär kärnfunktion**
  Den linjära kärnan representerar en linjär transformation av datan och ges av det inre produkten av de insatta variablerna. Den är lämplig för problem med linjära samband mellan variablerna.

- **Polynomisk kärnfunktion**
  Den polynomiska kärnan transformerar datan till en högre dimensionell egenskapsrymd med hjälp av polynomfunktioner. Den kan fånga icke-linjära samband med en grad av komplexitet som bestäms av den valda polynomgraden.

- **Radial basisfunktionskärna (RBF)**
  RBF-kärnan är ett av de mest populära valen för SVR. Den mappar datan till en oändligt dimensionell egenskapsrymd med hjälp av gaussiska funktioner. RBF-kärnan kan fånga komplexa icke-linjära samband och är lämplig när det inte finns någon förhandskunskap om datans underliggande struktur.

- **Sigmoids kärnfunktion**
  Sigmoidkärnan är ett annat alternativ som transformerar datan med hjälp av sigmoidfunktioner. Den kan vara användbar för vissa tillämpningar, men den används inte lika ofta som de andra kärnorna som nämnts ovan.

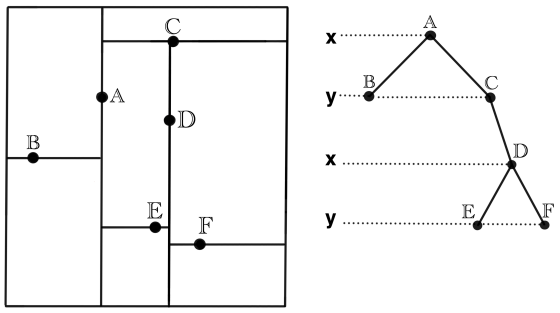
2.2.3 k-Nearest Neighbor Method & k-d Tree

kNN är en ML-metod som går ut på att skatta utdatan genom att titta på vad de närmaste punkterna i indatan har för utdata.[3] Eftersom det i vårt fall rör sig om ett regressionsproblem skattas utdatan med hjälp av att ta medelvärdet av utdatan från de k närmaste indatapunkterna i träningsdatan.

Att hitta de k närmaste datapunkterna är ett optimeringsproblem med flera lösningsmetoder, varav en av dem går ut på att bygga ett *k-d tree* - ett k-dimensionellt träd. Trädet är en datastruktur som effektivt lokaliserar närmaste datapunkten, eller "grannen", och har en struktur som liknar det binära sökträdet. Som beskrivet av S. Hanan (2006) [9], för att konstruera ett k-d träd börjar man med att slumpvis välja ut en startnod. I vårt 2-dimensionella exempel, med dimensionerna x och y, kallar vi startpunkten för A, som får bli vår startnod. Nästa steg är att jämföra nodens x-värde med nästa slumpvist vald punkt, låt oss kalla den för B. Eftersom B har ett mindre x-värde placeras B som barnnod på vänstra sidan om A. Nästa nod som läggs till kommer igen jämföras med A, och om noden har ett x-värde större än A hamnar noden på höger sida. Om noden har ett lägre x-värde kommer gå ner ett lager längs trädet, och låta punktens y-värde jämföras med y-värdet vid B, och placeras som barnnod på höger respektive vänster sida om det är större eller mindre. Med andra ord, varje nivå i trädet skiftar mellan att jämföra x- och y-värde. Denna

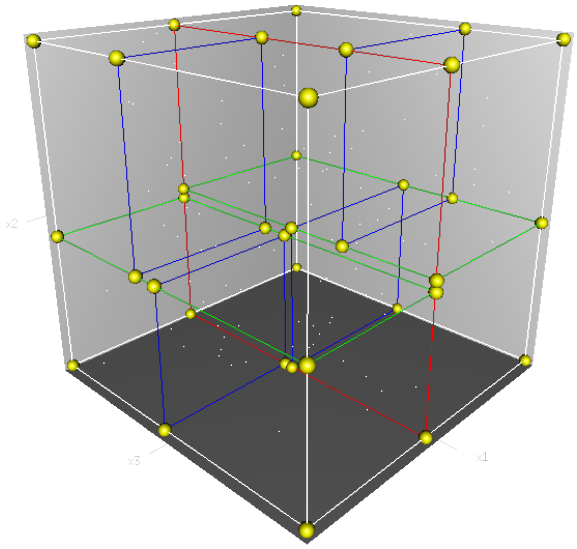
process är ekvivalent med att varje ny nod delar en yta i två, se figur 2.4. I de fall där punkterna befinner sig i rum av k dimensioner skiftar jämförelserna mellan alla k stycken variabler, och den visuella motsvarigheten är att varje nod skär genom rummet med "plan" av dimension $k - 1$.

Lokaliseringen av närmaste grannen sker sedan genom att man på samma sätt som beskrivet ovan navigerar sig ned genom trädet tills man når dess botten. Sedan jämförs närliggande noder i trädet för att testa vilken som ligger närmast. Lokaliseringen av närmaste grannen får således komplexiteten $\mathcal{O}(\log(n))$, där n är antalet noder i trädet. Insättningen av en ny nod får på grund av trädstrukturen en genomsnittlig komplexitet $\mathcal{O}(\log(n))$, och i värsta fall $\mathcal{O}(n)$. Skapandet av ett träd med totalt n noder har därmed komplexiteten $\mathcal{O}(n \log(n))$. Med detta menas att då indatan exempelvis dubbleras, $2n$, skalas den förväntade tiden algoritmen kommer ta $2n \log(2n)$.



Figur 2.4: Konstruktion av 2-dimensionellt träd

I figur 2.5 visas ett exempel på hur punkter och skärande plan skulle se ut i det 3-dimensionella fallet. Figuren visar inte exakt var punkterna är placerade, utan endast planen som visualiserar hur k-d trädet delar in rummet.



Figur 2.5: 3-dimensionellt k-d träd. Från wikimedia, Btyner, GNU General Public License [10]

2.2.4 Neurala Nätverk

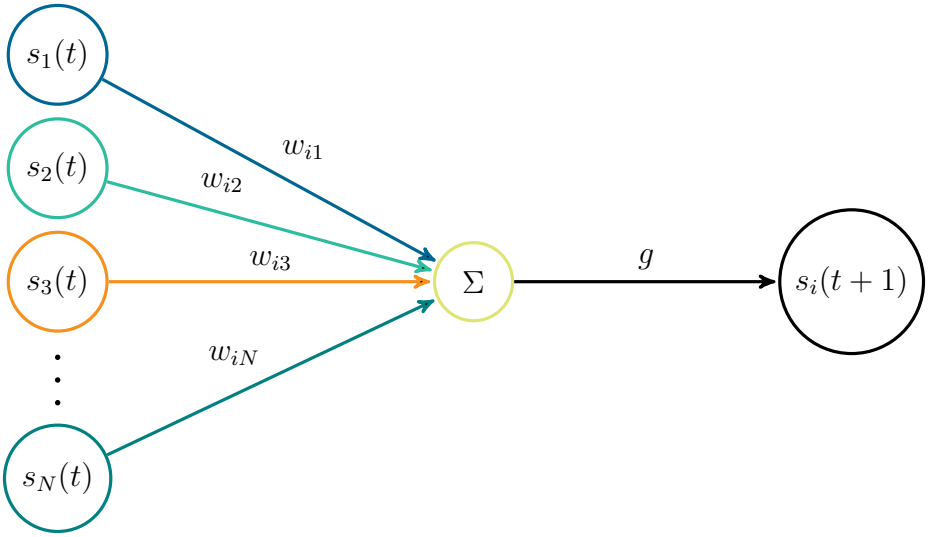
Neurala nätverk bygger på samma grundläggande koncept som kännetecknar hur den biologiska hjärnan fungerar. En hjärna består av små enheter som kallas neuroner, som är sammankopplade i ett nätverk. Varje neuron tar emot signaler från andra neuroner och ger en utgående signal till andra neuroner. När den digitala motsvarigheten av en neuron uppfanns kallades det först för perceptron [11], innan det antog namnet neuron.

Som visas i figur 2.6 tar neuronen in indata, summerar dessa med kanternas korresponderande vikter och adderar en bias, som sedan tas in av en så kallad aktiveringsfunktion som framställer den slutgiltiga utdatan. Summan beräknas enligt ekvation 2.27, som sedan tas in av aktiveringsfunktionen g i ekvation 2.28.

$$b_i(t) = \sum_{j=1}^{N} w_{ij} s_j(t) - \theta_i \tag{2.27}$$

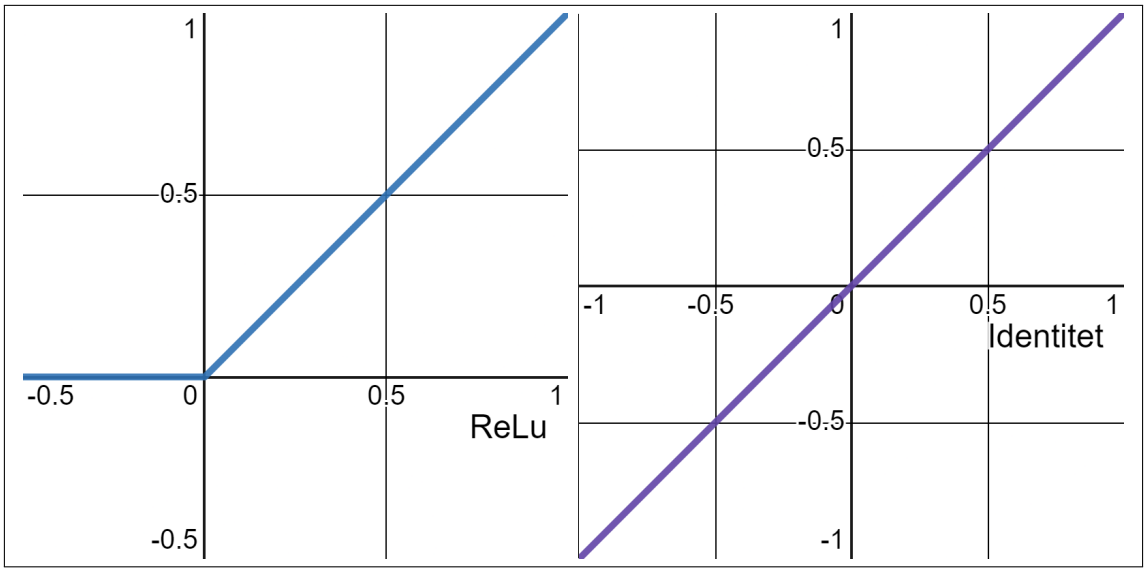
$$s_i(t+1) = g(b_i(t)) \tag{2.28}$$

Biasen θ adderas för att att ge nätverket en extra frihetsgrad till att justera utdatan oberoende av indatan. Utan biasen hade neuronen endast fått en linjärkombination av indatan, vilket skulle begränsa dess uttryckningsförmåga kraftigt. Aktiveringsfunktionen avgör om och hur varje neuron i ett neuralt nätverk blir aktiverad, baserat på dess utdata. Syftet är att bestämma om utdatan från neuronen är ”viktig” för att på lämpligt sätt aktivera den. I första hand reducerar aktiveringsfunktionen det linjära sambandet mellan in- och utdata, vilket ger utrymme för det neurala nätverket att lära sig mer invecklade förbindelser.



Figur 2.6: Neuron

Aktiveringsfunktionen avgör om och hur varje neuron i ett neuralt nätverk blir aktiverad, baserat på dess utdata. Syftet är att bestämma om utdatan från neuronen är ”viktig” för att på lämpligt sätt aktivera den. I första hand reducerar aktiveringsfunktionen det linjära sambandet mellan in- och utdata, vilket ger utrymme för det neurala nätverket att lära sig mer invecklade förbindelser. I Figur 2.7 visas aktiveringsfunktionerna $ReLu(x) = max\{0, x\}$, och $Identitet(x) = x$.



Figur 2.7: ReLu och identitetsfunktion

Ett neuralt nätverk är kort sagt en sammansättning av många sådana neuroner. Indatan till hela nätverket passerar genom så kallade dolda lager. Varje dolt lager består av ett visst antal neuroner som skickar sin utdata till nästa lager neuroner. Till sist har datan passerat och hanterats av alla neuroner i nätverket, och det slutgiltiga resultatet är producerat. Alla neuroners parametrar, vikter och bias, kan justeras för att nå bättreresultat, beroende på vad man optimerar efter. I figur 2.8

visas ett neuralt nätverk med endast ett dolt lager av neuroner.

Det finns flera typer av neurala nätverk, men det mest grundläggande är det så kallade feedforward-nätverket, som består av ett antal lager med noder. I ett sådant nätverk rör sig informationen framåt genom lager av neuroner från indata till utdata genom att vikterna mellan noderna justeras baserat på erfarenhet och feedback. Vikterna representerar styrkan på kopplingen mellan neuroner, och justeras för att nätverket ska kunna lära sig att känna igen mönster i data.

Under träningstiden matas nätverket med indata och dess förutsagda utdata jämförs med rätt svar. Träningsprocessen justerar sedan nätverkets vikter så att dess förutsägelser blir mer precisa över tiden. När nätverket har tränats tillräckligt på en given uppsättning data, kan det sedan användas för att göra förutsägelser på liknande data.

2.2.4.1 Gradientnedstigning, och Målfunktion

För att beskriva vad dessa begrepp innebär, måste vi först klargöra för vilken målfunktion det neurala nätverket optimerar efter. En målfunktion för neurala nätverk är *Minimum Squared Error Loss* (MSE). Målet är alltså att minimera felet, skillnaden mellan förutspådda värden och faktiska värden. *Sum Squared Residuals* (SSR) beskriver summan av kvadraten av alla fel, och MSE beskriver snittet på felet i kvadrat, så

$$SSR = \sum_{i=1}^{n} (y_{pred,i} - y_{real,i})^2, \tag{2.29}$$

$$MSE = \frac{1}{n} SSR. \tag{2.30}$$

Notera att med denna målfunktion straffar större fel hårdare än små fel, eftersom differensen kvadreras.

Nästa steg är att redogöra för hur man uppfyller målet effektivt. Gradientnedstigning, på engelska "gradient descent", går ut på att undersöka vilka ändringar bland parametrarna som ger en minskning i målfunktionen, och sedan ändra på dessa parametrar ett litet steg i den riktningen. Denna process upprepas sedan iterativt tills man nått tillsfredsställande resultat, eller inte kan hitta en förbättring.

När man utför gradientnedstigning på ett nätvärk uppdateras alla vikter och biaser som utgör nätvärkets parametrar p enligt:

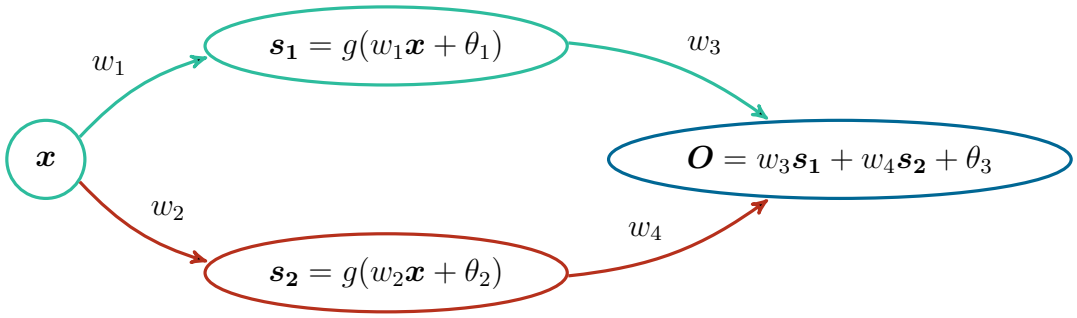
$$p' = p - \alpha \frac{\partial f_l}{\partial p}, \tag{2.31}$$

där p' är det uppdaterade värdet på parametern, α är steglängden, och f_l är målfunktionen.

Stokastisk gradientnedstigning (SGD från engelskans *Stochastic Gradient Descent*) fungerar som vanlig gradientnedstigning, men med tillägget att modellens parametrar uppdateras iterativt på godtyckligt samplade delmängder av träningsdatan. Den godtyckliga samplingen är vad som gör algoritmen "stokastisk". Gradienten till målfunktionen beräknas för varje sådan delmängd, och parametrarna justeras i riktningen där gradienten har negativt värde. Användningen av SGD resulterar i lägre beräkningskostnader, samt att den stokastiska egenskapen kan hjälpa optimeringsalgoritmen att undvika lokala optimum och hitta bättre lösningar. Nackdelen är att den slumpmässiga karaktären hos uppdateringarna också medför större svängningar och instabilitet i jämförelse med andra optimeringsalgoritmer [12].

2.2.4.2 Exempel på ett neuralt nätvärk

För att visa hur det fungerar i praxis kan det byggas ett enkelt neuralt nätvärk, som visas i figur 2.8.



Figur 2.8: Ett exempel på ett enkelt neuralt nätverk

En liten uppsättning träningsdata matas till nätverket, tre par av in- och utdata, kan se ut enligt följande:

$$\{(x_1, y_1), (x_2, y_2), (x_3, y_3)\}.$$

Sedan passerar dessa värden genom för att få de nätverkets förutspådda värden

$$\{y_{pred,1}, y_{pred,2}, y_{pred,3}\},$$

vilket ger oss målfunktionen

$$MSE = \frac{1}{3}\sum_{i=1}^{3}(y_{pred,i} - y_i)^2.$$

Genom att traversera det neurala nätverket baklänges, från utdata till indata, ser

man att $y_{pred,i}$ ges av

$$\begin{aligned} y_{pred,i} &= s_{1,i}w_3 + s_{2,i}w_4 + \theta_3 \\ s_{1,i} &= g(b_{1,i}) \\ s_{2,i} &= g(b_{2,i}) \\ b_{1,i} &= x_i w_1 + \theta_1 \\ b_{2,i} &= x_i w_2 + \theta_2 \\ i &\in \{1, 2, 3\} \end{aligned}$$

För att uppdatera nätvärket körs sedan gradientnedstigning på alla parametrar p enligt ekvation 2.31. Om vikten w_2 till exempel uppdateras, räknas den partiella derivatan ut via kedjeregeln, som visas i ekvation 2.32.

$$\begin{aligned} \frac{\partial}{\partial w_2}(MSE) &= \frac{1}{3} \cdot \frac{\partial}{\partial w_2}(SSR) \\ &= \frac{1}{3} \cdot \frac{\partial SSR}{\partial y_{pred,i}} \cdot \frac{\partial y_{pred,i}}{\partial s_{2,i}} \cdot \frac{\partial s_{2,i}}{\partial b_{2,i}} \cdot \frac{\partial b_{2,i}}{\partial w_2} \\ &= \frac{1}{3} \cdot \sum_{i=1}^{3} 2(y_{pred,i} - y_i) \cdot w_4 \cdot g'(b_{2,i}) \cdot x_i \end{aligned} \tag{2.32}$$

3
Metod

Metodavsnittet beskriver tillvägagångssättet och procedurerna som vidtagits för att genomföra projektet och uppnå studiens mål. Detta avsnitt ger en detaljerad beskrivning av forskningsdesignen, metoder för data-anpassning och verktyg som används i studien. Delar av den Kod som använts kommer presenteras i begränsad pseudo struktur, för komplett kod se billaga C.

3.1 Data från Direkt Numerisk Simulering

För att möjliggöra användning av ML behövs korrekt data. I denna implementation kommer DNS data användas för två olika strömnings fall: Small wave och large wave. Datan är framtagen av L. Davidsson [13] I DNS datan fås korrekta värde på: X_c, Y_c, p, $\overline{u}$, $\overline{v}$, $\overline{u'u'}$, $\overline{v'v'}$, $\overline{w'w'}$, $\overline{u'v'}$ och ε. Från dessa kan $||S_{ij}||$ räknas ut enligt ekv 3.1 och 3.2. För överblick av de två fallen se bilaga A.1 för *small wave* och bilaga B.1 för *large wave*

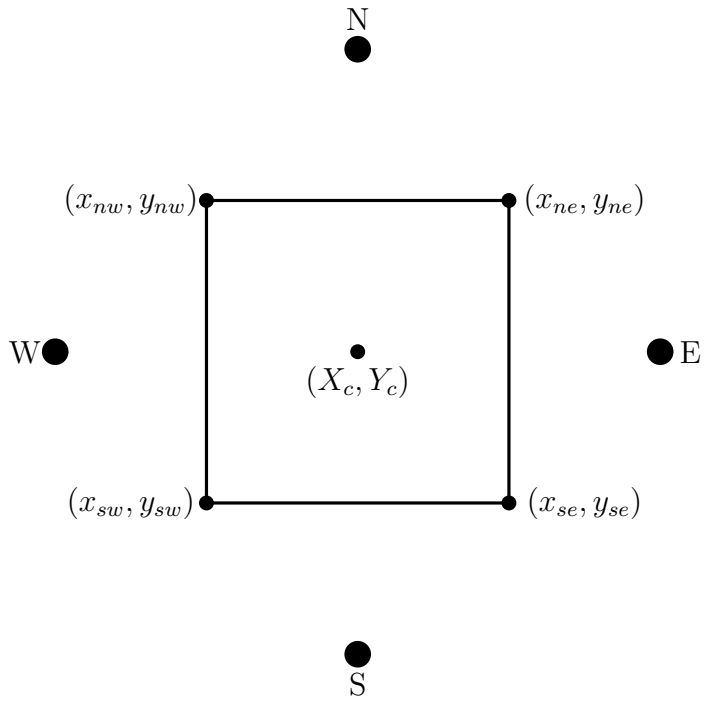
$$S_{ij} = \frac{1}{2}\left(\frac{\partial \overline{v}_i}{\partial x_j} + \frac{\partial \overline{v}_j}{\partial x_i}\right) \tag{3.1}$$

$$||S_{ij}|| = \left(\sqrt{\left(\frac{\partial \bar{u}}{\partial x}\right)^2 + \frac{1}{2}\left(\left(\frac{\partial \bar{u}}{\partial y}\right)^2 + 2\frac{\partial \bar{u}}{\partial y}\cdot\frac{\partial \bar{v}}{\partial x} + \left(\frac{\partial \bar{v}}{\partial x}\right)^2\right) + \left(\frac{\partial \bar{v}}{\partial y}\right)^2}\right) \tag{3.2}$$

Ytterliggare ett fall med simpel kanalströmning kommer testas där datan är framtagen av M.Lee och R.Moser [14]. I den datan ingår: y^+, $\overline{u}$, $\overline{w}$, P, $\frac{\partial \overline{u}}{\partial y}$, $\overline{u'u'}$, $\overline{v'v'}$, $\overline{w'w'}$, ε.

3.1.1 Meshgrid och hastighetsgradienter

För att beräkna gradienterna behöver ett mesh(rutnät) spännas upp genom att ta fyra punkter och dess hastigheter.



Figur 3.1: Exempel av en ruta i de rutnät som anänts

Datan innehåller ett existerande meshgrid med positioner och hastigheter för flera punkter i flödet, t.ex. (x_{sw}, y_{sw}), (x_{se}, y_{se}), (x_{nw}, y_{nw}) och (x_{ne}, y_{ne}). Det tillåter en att spänna upp ett meshgrid för hastighetsgradienterna och beräkna dem med hjälp av ekvation 3.3. Låt v_1 vara hastigheten mitt i cellen.

$$v_{1,P} = \frac{1}{4}(v_{1,sw} + v_{1,se} + v_{1,nw} + v_{1,ne}) \tag{3.3}$$

Gradienterna kan med hjälp av Greens formula uttryckas som

$$\begin{aligned}\frac{\partial \overline{v_1}}{\partial x_1} &= \frac{1}{V}\int_A v_1 n_1 \, dA \\ &= \frac{1}{V}\sum_{i=e,n,w,s}(v_1 n_1 A)_i \\ &= \frac{1}{V}\left((v_1 n_1 A)_e + (v_1 n_1 A)_n + (v_1 n_1 A)_w + (v_1 n_1 A)_s\right)\end{aligned} \tag{3.4}$$

där hastighetsgradienten $\frac{\partial \overline{v_1}}{\partial x_1}$ mitt i cellen.

3.2 Maskininlärning - Support Vector Regression

Flödet beskrivs väl av gradienterna och $||S_{ij}||$, dermed är de ett naturligt val av träningsdata för SVR .

Koden 3.1 påvisar den övergripande metodiken för den aktuella tillämpningen av SVR. Det som görs är att initialt defineras målvariabeln, i detta fall C_my_DNS. Detta är C_μ framtagen enligt ekvation 2.9 med hjälp av DNS-data. Därefter skapas en skalad version av datan vid hjälp av $StandardScaler()$ som skalar indatan genom standardisering. Därefter defineras de oberoende variablarna som i detta fall valts till $\frac{\partial u}{\partial x}, \frac{\partial v}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial v}{\partial y}, ||S_{ij}||$.

```
Y = df_SW['cmy']

df_SW_scaled = pd.DataFrame(scaler.transform(df_SW),
               columns=df_SW.columns)

X = df_SW_scaled[['dudx','dvdx','dudy','dvdy','duidxj']]

model = SVR(kernel='rbf', C=1, epsilon=0.01)
svr = model.fit(X, Y)

pickle.dump(svr, open('trained_models/svr.pkl','wb'))
joblib.dump(scaler, 'scaler.save')
```

Listing 3.1: ML metodik

Till sist initieras modellen, i detta fall SVR med radial basisfunktion som kärna vilket är viktigt då det önskade sambandet inte kan antas vara linjärt beroende. Polynomisk kärnfunktion kan också användas för att modellera inte-linjära sammanhäng, men används inte i detta projektet eftersom körtiden blev oacceptabelt lång.

Avslutningsvis tränas modellen med datan som framtagits och sparas undan tillsammans med skaleringsparametrar för senare testing.

Efter dessa steg kvarstår att testa modellen som nu skapats. Detta görs på ny data, där processen återupprepas. Däremot skapas inte en ny modell för den nya datan, istället används följande kommando.

```
C_my_prediction = model.predict(X_new_case)
```

Där X_new_case tagits fram med samma metodik som tidigare.

3.2.1 Optimering med Support Vector Regression

För att förbättra resultatet på den framtagna modellen från SVR fanns det flera sätt. Först ändrades hyperparametrarna ϵ och C i en iterativ process där felet och framtagna grafer analyserades efter varje ändring tills ett optimalt resultat uppnåts. Det valdes även att testa begränsa träningsdatan utifrån kriteries att $S_{ij} > 15$. Det säkerställer att modellen tränas på de område i strömningen som har hög turbulence.

3.3 Skattning via k-d Tree

En metod som kan användas för att skatta utdatan, C_μ, är kNN-metoden, enligt beskrivningen i avsnitt 2.2.3. Detta alternativ prövas eftersom SVR kan vara en långsam metod, speciellt på stora datamängder. Det är av intresse att jämföra de två metoderna och se vilken som presterar bäst. Detta innebär att istället för att ha en modell som approximerar C_μ, kommer direkt simulerade värden användas. I det aktuella programmeringsspråket Python, som används i detta arbete, finns objektet *KDTree* tillgängligt via scipy-biblioteket för att underlätta implementeringen.

Metoden testas både på att skatta värden på C_μ på samma fall, som Small Wave, och från fall till fall, exempelvis Large Wave till Small Wave. Utöver träffsäkerheten i algoritmens skattningar kommer även träningstiden jämföras med övriga modeller. Huruvida det tar signifikant längre tid att träna SVR och neurala nätverk i jämförelse testas också.

I Listing 3.2 visas hur kNN-regressionen implementeras i Python.

```
# kNN regression for C_my with k = 10

# Create a list of the k nearest neighbors for each point
k = 10
knn = c_my_tree.query(test_points, k=k)
# Distances between each point and its k nearest
   neighbors
distances = knn[0]
indices = knn[1]
# C_my values of the k nearest neighbors
knn_c_my = training_c_my[indices]
average_knn_c_my = np.mean(knn_c_my, axis=1)
# Standard deviation
std_knn_c_my = np.std(knn_c_my, axis=1)
# Ignore the points that have a standard deviation
   greater than 0.5
std_threshold = 0.5
average_knn_c_my = np.ma.masked_where(std_knn_c_my >
   std_threshold, average_knn_c_my)
```

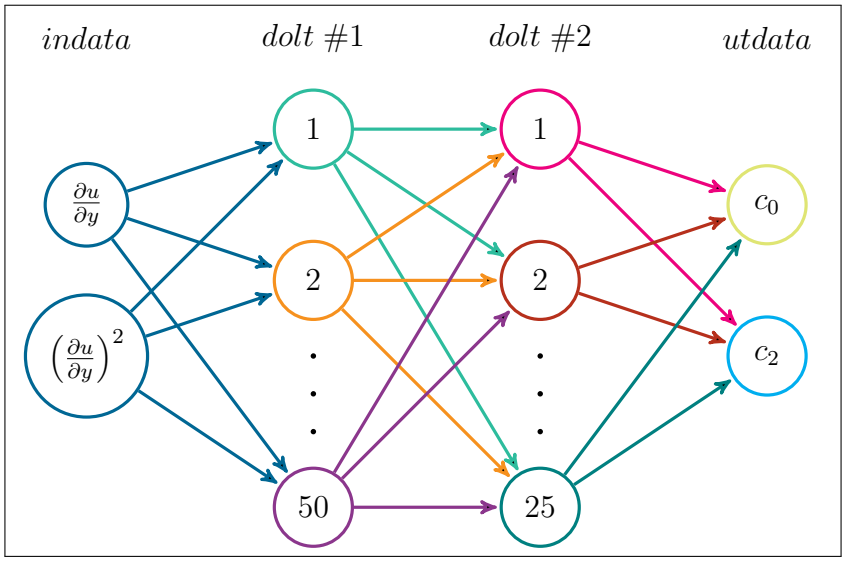
Listing 3.2: Kod för kNN

3.4 Neurala Nätverk

Då SVR kan inte skatta flera variabler samtidigt undersöks hur neurala nätverk kan skatta c_0 och c_2. Det neurala nätverket skildras i figur 3.2, som beskriver hur vi använder gradienterna $\frac{\partial \overline{u}}{\partial y}$ och $\left(\frac{\partial \overline{u}}{\partial y}\right)^2$ som indata för att generera utdatan c_0 samt c_2.

Anledningen till att vi använder just den indatan beror på att datan på resterande gradienter är satta till noll i fallet vi modellerar, kanalströmming. En modell som endast inkluderade termen $\left(\frac{\partial \overline{u}}{\partial y}\right)^2$ testades och visade sig fungera acceptabelt. Emellertid genererade modellen i vissa fall den additiva inversen av korrekta spänningar, vilket löstes med hjälp av den extra indatapunkten $\frac{\partial \overline{u}}{\partial y}$. Detta eftersom att då gradienten kvadreras kan inte längre riktningen tas hänsyn till. Det är även anledningen till att gradienten $\frac{\partial \overline{u}}{\partial y}$ lades till som indata, då denne tar hänsyn även till riktning.

Anledningen till att inte geometriska variabler användes som indata är att de inte är generaliserbara. Modellen blir snabbt överpassad till det specifika fallet. Gradienter och liknande variabler är däremot mer generella mellan diverse strömmningsfall och tillåter därför en mer generell modell. Dessutom användes inte Reynoldsspänningar som indata då slutmålet är att modellera dessa med hjälp av utdatan från nätverket. Modellen blir därmed meningslös då det som modelleras krävs för modelleringen.



Figur 3.2: Neuralt nätverk som skattar c_0 och c_2

Detta genomfördes med hjälp av python-biblioteket *pyTorch* som förenklar arbete med neurala nätverk väsentligt. Det som behöver genomföras i kodväg är att skala den givna datan, detta görs med hjälp av biblioteket *skLearn* och dess *StandardScaler* funktion. Därefter delas datan i träningsset och valideringsset. Samtlig data som krävs placeras därefter i tensorer och skickas till nätverket. Då nätverket tränats på cirkus 80% av den givna datan kan en prediction göras på resterande data. Resultatet kan därefter illustreras, detta beskrivs mer i avsnitt 4.3. I figuren 3.2 illustreras det nätverk som användes i detta arbete. Det består av 2 stycken noder för indata, därefter 2 stycken dolda lager bestående av 50 respektive 25 noder. Till sist ett lager med 2 stycken noder för de båda önskade utdata-vektorerna. För båda dolda lager används $ReLu = max\{0, b_i\}$ som aktiveringsfunktion, medan utdata som skall förutspå kontinuerliga värden använder identitetsfunktionen.

3.5 Utvärdering

För att utvärdera de framtagna modellerna kommer ekvation 3.5 användas. Det framräknade felet kommer sedan jämföras med felet som fås om standard värde $C_\mu = 1$ används.

$$Fel \equiv \frac{std(C_\mu - C_\mu^{DNS})}{\sqrt{mean(C_\mu^2)}} = \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}((C_{\mu,i} - C_{\mu,i}^{DNS}) - \frac{1}{N}\sum_{j=1}^{N}(C_{\mu,j} - C_{\mu,j}^{DNS}))^2}}{\sqrt{\frac{1}{N}\sum_{k=1}^{N}(C_{\mu,i}^2)}} \tag{3.5}$$

Där N är antalet datapunkter i de båda vektorerna C_μ samt C_μ^{DNS}. Notera att denna felfunktion appliceras först efter att modellerna har tränats. Det är inte samma felfunktion som modellerna är optimerade för att minimera, som MSE. Notera även att felet beräknas som en variationskoefficient istället för exempelvis *Root Mean Square* (RMS). Detta på grund av att felet önskas i procentuell form och inte i den aktuella enheten.

Dessutom är det värt att notera att i fallet med det neurala nätverket beräknas inte felet direkt mellan indatan och utdatan från nätverket. Istället beräknas felet mellan samtliga Reynoldsspänningar tagna från DNS data för det aktuella fallet och Reynoldsspänningar beräknade med utdatan från det neurala nätverket.

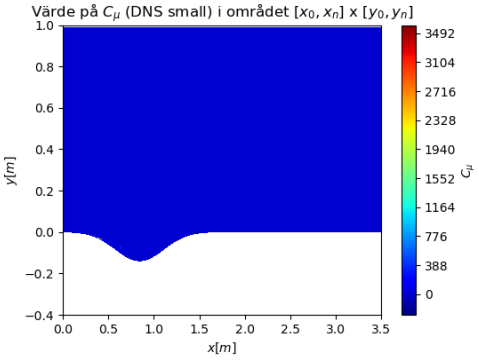
4

Resultat

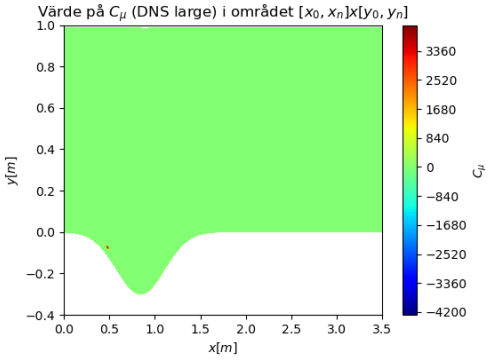
Resultatavsnittet i denna kandidatuppsats presenterar de fynd och resultat som härleds från den genomförda arbetet. Resultan utgör fundamentet för efterföljande analys, tolkning och diskussion i de följande avsnitten av uppsatsen.

4.1 Support Vector Regression

I figurerna nedan visas C_μ som erhållits direkt från DNS i sin respektive domän. I figur 4.1 och 4.2 syns ingen variation på C_μ, vilket beror på en del extrema och ofysikaliska värden.

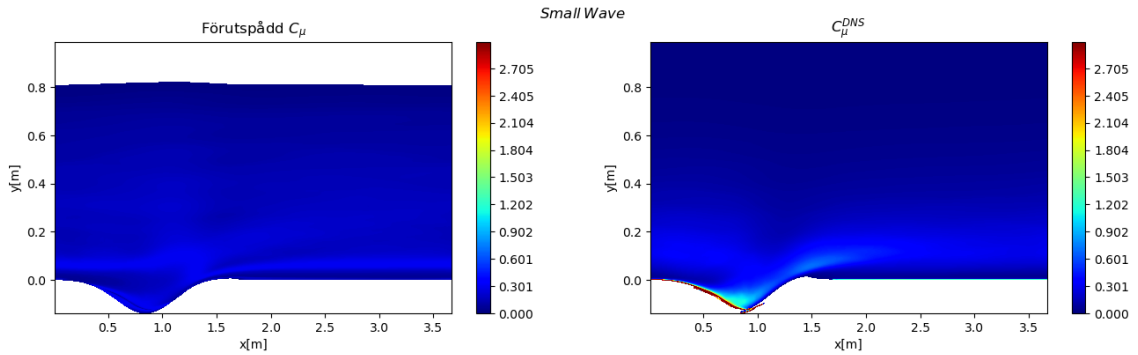


Figur 4.1: DNS C_μ för "Small Wave"

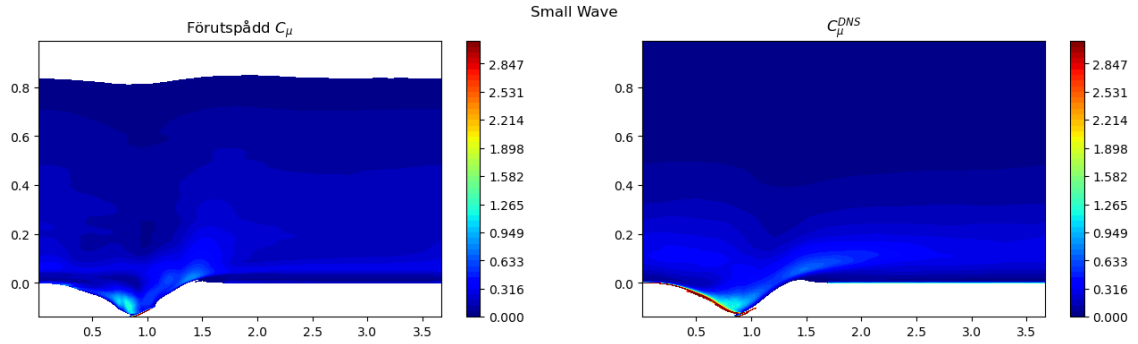


Figur 4.2: DNS C_μ för "Large Wave"

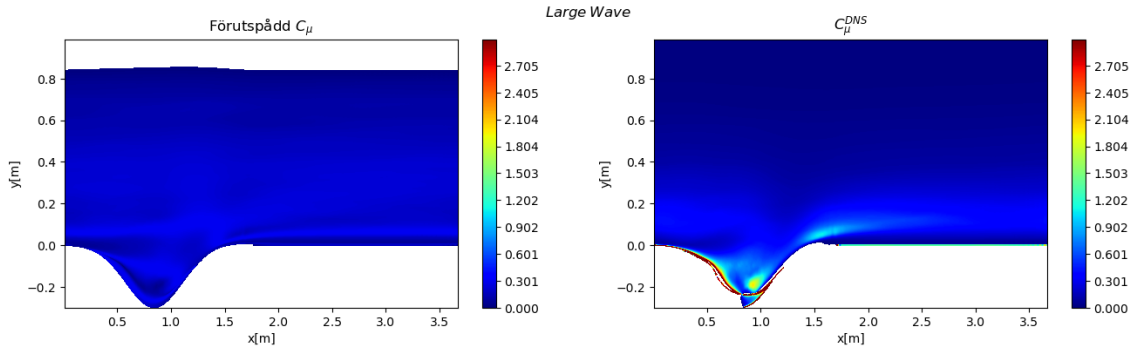
I figur 4.3-4.6 syns förutspådd C_μ från SVR samt C_μ från DNS, båda begränsade inom intervallet 0-3 vilket tar bort extrempunkterna och visar variationen bättre. Punkter utanför intervallet kommer inte ha någon färg. SVR modellen som förutspått C_μ är tränad på *small wave*. Figurerna visualiserar hur väl modellen förutspår värden och vilka områden i domänen den presterar bättre eller sämre på.



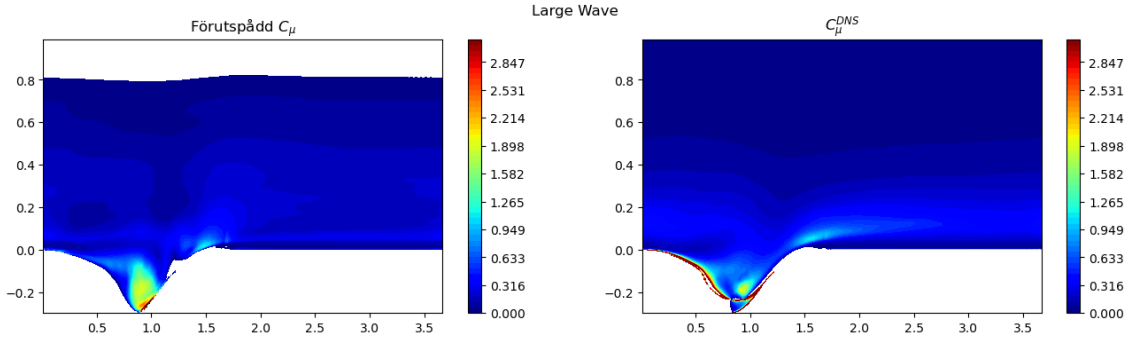
Figur 4.3: Förutspådd C_μ och DNS, indata: $||S_{ij}||$



Figur 4.4: Förutspådd C_μ och DNS, indata: $\frac{\partial u}{\partial x}, \frac{\partial v}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial v}{\partial y}, ||S_{ij}||$



Figur 4.5: Förutspådd C_μ och DNS, indata: $||S_{ij}||$



Figur 4.6: Förutspådd C_μ och DNS, indata: $\frac{\partial u}{\partial x}$, $\frac{\partial v}{\partial x}$, $\frac{\partial u}{\partial y}$, $\frac{\partial v}{\partial y}$, $||S_{ij}||$

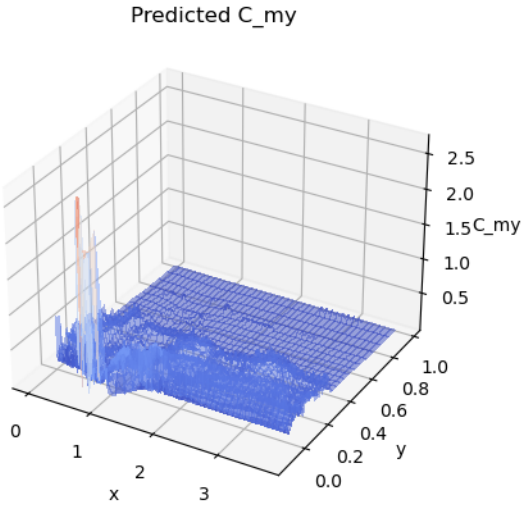
Felen mellan förutspådda värden och faktiska värden räknades ut som beskrivet i metoddelen. Felen visades vara orimligt stora och presenteras därför inte numeriskt.

4.2 k-Nearest Neighbours Regression

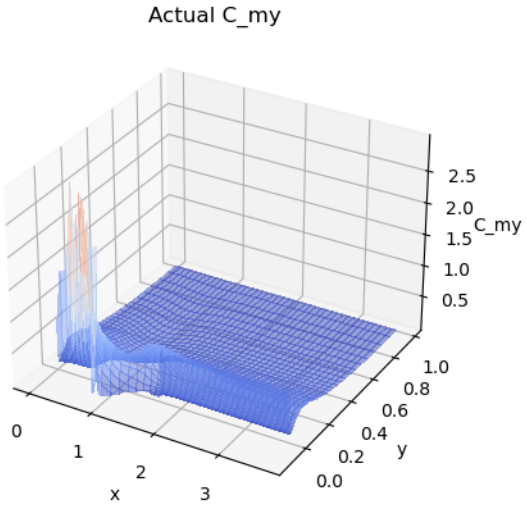
Skapandet av ett k-d träd och utförandet av kNN-regressionen tar inte mer än ett par sekunder. Emellertid försvagas träffsäkerhet och stabilitet. Som visas i figurerna 4.7 och 4.8 följer de skattade C_μ-värdena de faktiska värdena ganska bra, men buller förekommer också i stor grad. k-d trädet är byggt på de partiella derivatorna $\frac{\partial u}{\partial y}$ och $\frac{\partial v}{\partial x}$, som beräknas från DNS-datan erhållen av *large wave*-fallet, och testas på fallet *small wave*.

Tabell 4.1: Tabell över felen *Mean Average Error* (MAE), *Mean Squared Error* (MSE), *Root Mean Squared Error* (RMSE) och *Mean Average Percentage Error* (MAPE).

Felberäkningsmetod	**Fel**
MAE	0.080
MSE	0.033
RMSE	0.182
Variationskoefficient, se 3.5	0.725
MAPE	57.57%



Figur 4.7: Förutspådd C_μ från kNN regression



Figur 4.8: C_μ från DNS för liten våg

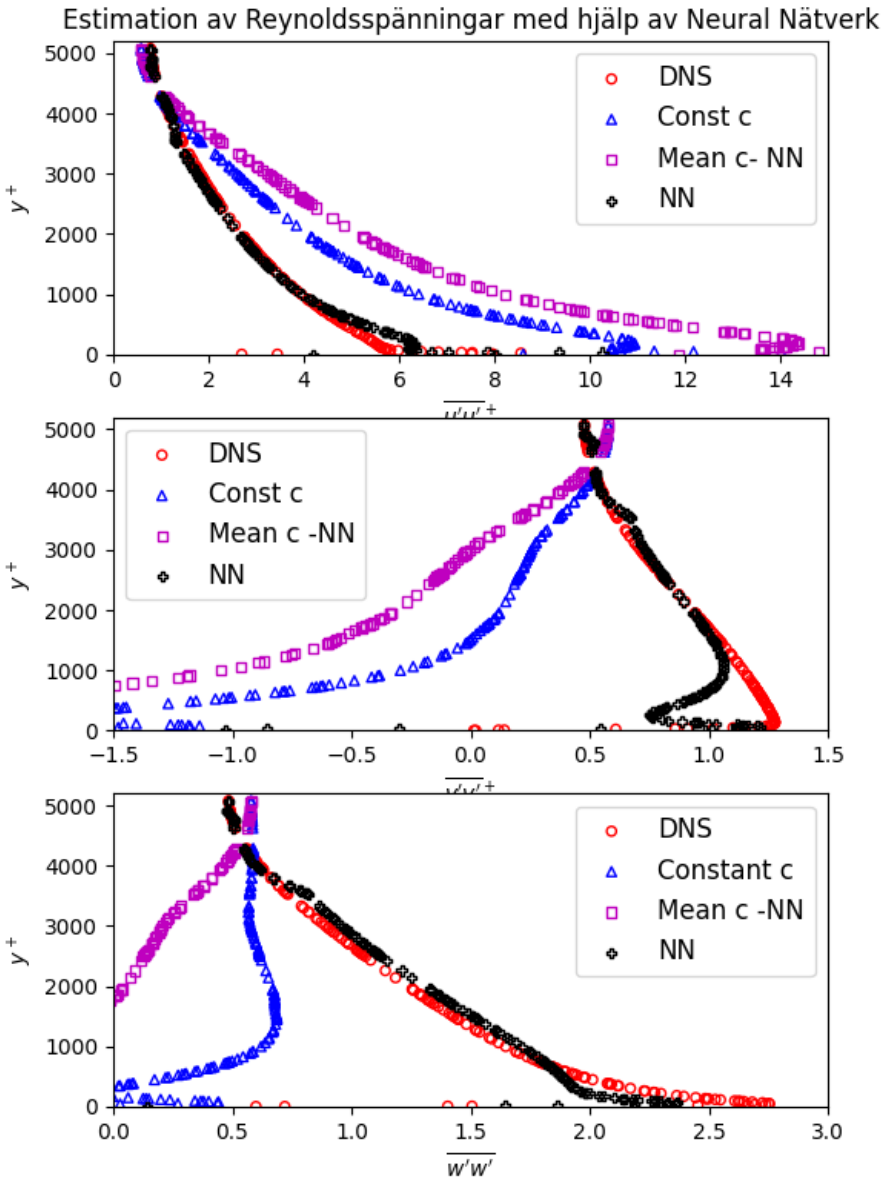
4.3 Neuralt Nätverk

Resultatet från arbetet med neurala nätverk kan ses i figure 4.9. Det är tydligt att modellen fungerar väl för Reynoldsspänningarna $\overline{u'u'}$, $\overline{w'w'}$ och mindre väl för $\overline{v'v'}$. Modellen fungerar även bättre än de vedertagna konstanterna från [6]. Dock visar medelvärdet från modellen sämre prestation, vilket antyder att det finns värden som modellen genererar som inte är tillförlitliga. Detta kan bero på att spänningarna går mot stora värden i väggen. För att lösa detta problem valdes att ignorera dessa värden och utföra beräkningar endast för punkter där $y^+ > 30$, vilket framgår i figur 4.9.
Resultatet av detta blir att felet inte representerar de faktiskt intressanta resultatet i figur 4.9. Felet, enligt ekvation 3.5, blev ≈ 1 för hela domänen, vilket uppenbarligen inte gäller i de intressanta området, $y^+ > 30$, då modellen presterar bättre och bättre då avståndet, y^+, ökar. Istället filtrerades alla värden som låg så pass nära väggen att de tydligt ”spårat ur” bort. Detta gav upphov till följande fel. Anledningen till att just $y^+ > 30$ är intressant är på grund av att i intervallet $0 < y^+ < 30$ är det främst viskösa krafter som dominerar och ingen turbulens finns i väsentlig utsträckning.

$$
\begin{aligned}
Fel \quad & \overline{u'u'} = 0.13516095767438668 \\
Fel \quad & \overline{v'v'} = 0.1931947039337504 \\
Fel \quad & \overline{w'w'} = 0.12473556709800085
\end{aligned}
$$

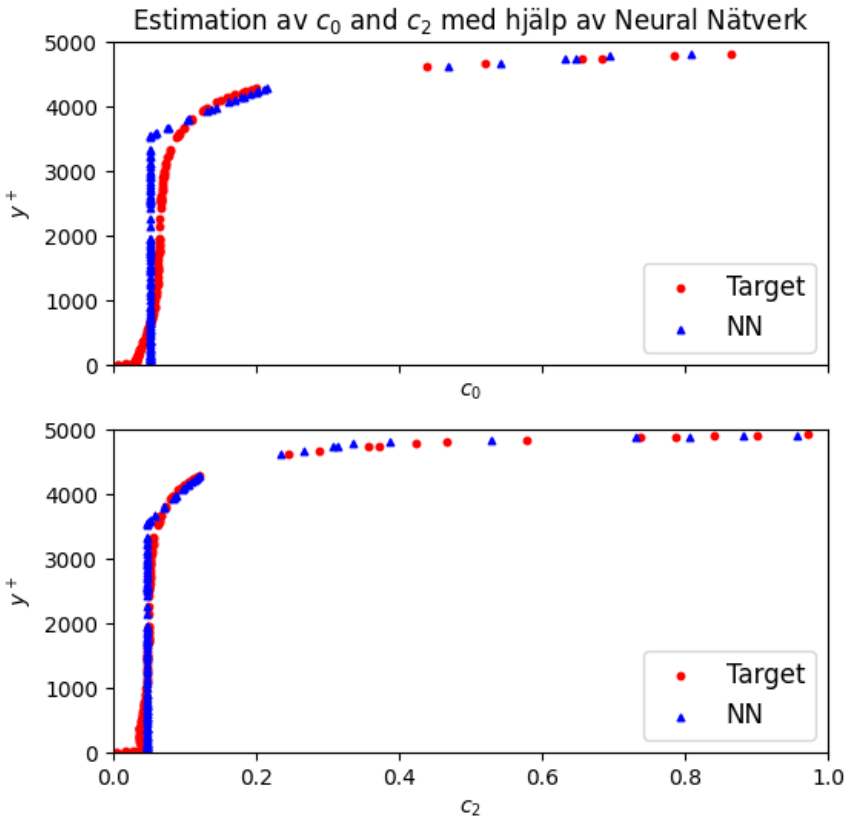
Dessa kan sedan jämföras med felet som de vedertagna konstanterna [6] ger upphov till.

$$
\begin{aligned}
Fel \quad & \overline{u'u'} = 0.30118036902243606 \\
Fel \quad & \overline{v'v'} = 1.2724279994778185 \\
Fel \quad & \overline{w'w'} = 1.568855674459809
\end{aligned}
$$



Figur 4.9: Estimation av Reynoldsspänningar i kanalströmmning framtaget med hjälp av neuralt nätverk.

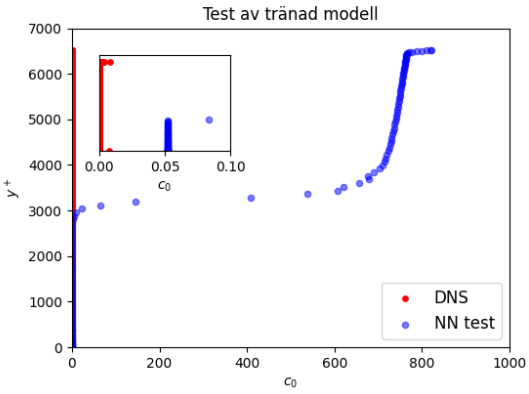
En ren jämförelse av vad det neurala nätverket har som indata jämfört med dess resultat kan ses i figur 4.10. Detta är ytterligare en illustration av samma model som visats i figur 4.9.



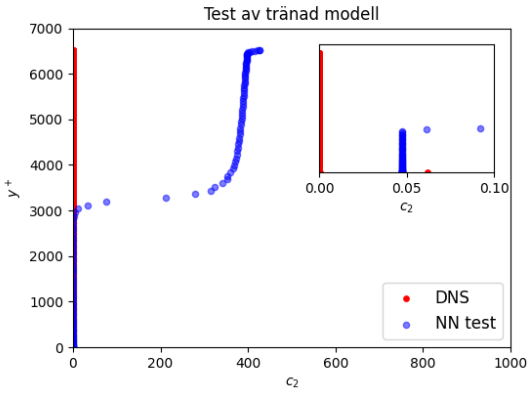
Figur 4.10: Estimation av c_0 och c_2 i kanalströmmning framtagna med hjälp av neuralt nätverk, ospecifierade axlar är desamma som de korresponderande specifierade axlarna.

4.3.1 Test med annat strömmningsfall

Testet med ett annat strömmningsfall resulterade i ytterst dåliga förutsägelser från modellen. Felet blir uppenbart så pass stort att det inte är meningsfullt att beräkna det enligt 3.5 eller liknande.



Figur 4.11: Resultat från test av neuralt nätverk för c_0, ospecifierade axlar är desamma som de korresponderande specifierade axlarna.



Figur 4.12: Resultat från test av neuralt nätverk för c_2, ospecifierade axlar är desamma som de korresponderande specifierade axlarna.

5
Diskussion

Diskussionsavsnittet ger möjlighet till en kritisk analys och tolkning av resultaten i relation till arbetets syfte och problemställning. Syftet med avsnittet är att noggrant utforska implikationerna och betydelsen av resultaten. Dessutom hanteras eventuella begränsningar, inkonsekvenser eller oväntade resultat, vilket ger insikter om tillförlitligheten och validiteten hos resultaten. Genom en balanserad och nyanserad diskussion uppmuntrar detta avsnitt till reflektion, syntes och formulering av slutsatser som bidrar till kunskapsutvecklingen inom ämnesområdet.

5.1 Support Vector Regression

Enligt resultatet i kapitel 4.1 framgick det tydligt att SVR inte genererade noggranna värden, och felet var betydande. När endast variabeln $||S_{ij}||$ användes blev felet särskilt stort, och när fler invariabler inkluderades förbättrades resultatet endast marginellt. Dessutom förekom en del extrema och orimliga värden på C_μ som genererats av DNS-datan, som SVR inte kunde hantera utan en del besvär. Till exempel observerades negativa värden, samt värden flera tusen gånger högre än väntat. I $k-\omega$ modellen görs antagandet $C_\mu = 1$, och därmed bör inte C_μ avvika från detta värde med mer än 1, vilket innebär att det förväntade intervallet är $C_\mu \in [0, 2]$.
På grund av detta blev det svårt att utvärdera om SVR var en bra metod att använda.

5.1.1 Varför C_μ inte gick att optimera.

Efter felsökning i teorin upptäcktes det att på grund av att Boussinesq-antagandet, som använts i härledningen av formlerna som används i detta arbete, krävs endast att kvoten i ekvation 5.1 stämmer.

$$\nu_t = \frac{k}{\omega} C_\mu, \;\; C_\mu = 1 \tag{5.1}$$

Detta medför att både k samt ω kan anta ofysikaliska värden, medan kvoten bibehåller rimlig status. På grund av detta riskeras ofysikaliska värden för det slutgiltiga resultatet. Försök gjordes på att bearbeta datan innan uträkning men det blev en för stor del av datan som behövde korrigeras för att ge tillförlitligt resultat.

5.2 k-Nearest Neighbors Regression

Att skapa ett k-d träd på all relevant data tar inte mer än ett par sekunder, i kontrast till SVR som kan ta många minuter för större datamängder. Emellertid ger inte kNN-regressionen lika träffsäkert resultat, och producerar mycket buller i jämförelse med SVR-regressionens släta kurvor.

Eftersom metoden tar ett medelvärde av datapunkter direkt från C_μ erhållet från DNS, borde inte kNN-regressionen producera orimliga värden så länge träningsvärdena är rimliga. Ändå producerade modellen en del extrema förutspådda värden, även då de extrema värden för C_μ från DNS inte inkluderas i träningen. Orsaken är oklar, men de mest extrema förutspådda värdena hanterades enkelt genom att inte tillåta värden som ligger utanför ett visst avstånd från standardavvikelsen för alla förutspådda C_μ.

Till en början skapades k-d trädet på 80% av datan, och regressionen utfördes på resterande 20%. Detta fungerade bra, men att illustrera resultatet blev invecklat på grund av problem med datamanipulationen. Genom att träna på ett fall och testa på ett annat, exempelvis träna på *small wave* och testa på *large wave*, undveks dessa problem.

5.3 Neuralt Nätverk

Till att börja med fungerar det att optimera sig av denna ickelinjära *Eddy-viscousity* modell. Detta trots att C_μ inte kunde optimeras. Detta eftersom att då den ickelinjära *Eddy-viscousity* modellen inte bygger på $k-\omega$, vilket gör att k inte behöver modelleras. Således kan konstanterna därmed modelleras.

Resultatet från arbetet med neurala nätverk visar sig vara lovande. Detta visar figur 4.10 samt 4.9 som påvisar att denna modell presterar bättre än de tidigare vedertagna konstanterna $c_0 = c_1 + c_3 = 0,16$ och $c_2 = 0,11$ [6]. Detta gäller dock endast då man låter dessa konstanter variera längs y^+. Det vill säga, då $c_0 = f\left(\left(\frac{\partial \bar{u}}{\partial y}\right)^2, \frac{\partial \bar{u}}{\partial y}\right)$ samt $c_2 = f\left(\left(\frac{\partial \bar{u}}{\partial y}\right)^2, \frac{\partial \bar{u}}{\partial y}\right)$. En bidragande faktor till detta är att DNS data använts vid beräkningen av Reynoldsspänningarna. Konstanterna från [6] är inte anpassade för användning i kombination med DNS-data enligt [13], vilket resulterar i deras dåliga prestation.

Detta visar sig däremot endast gälla för kanalströmmningen. Då en annan typ av kanalströmmning testas i figur 4.11 och 4.12 presterar modellen ytterst dåligt. Detta kan bero på att i det nya fallet gäller det att $\frac{\partial}{\partial x} \neq 0$. Därmed blir strömmningen annorlunda och således c_0 och c_2. Detta blir tydligt då konstanterna analyseras i de bägge fallen. I det initiala fallet har konstanterna storleksordningen 10^{-1} till 10^{-2} för alla punkter i strömmningen, medan i det andra fallet har de storleksordningen 10^{-5} till 10^{-9}. De två fallen kan därmed vara för olika för att modellen ska ha möjlighet

att prestera väl för de båda samtidigt.

En annan möjlig förklaring är att modellen är överanpassad, vilket innebär att den är för specifierad på det första fallet och kan därmed inte hantera andra strömmningsfall. Dessutom är det möjligt att båda de tidigare nämnda förklaringarna bidrar till den bristande prestandan parallellt.

6
Slutsatser

Slutsatsavsnittet i denna kandidatuppsats fungerar som en avslutande reflektion över den genomförda forskningen och ger en sammanfattning av de viktigaste fynden och insikterna som erhållits under studien. Dessutom möjliggör detta avsnitt en kritisk utvärdering av de begränsningar och svårigheter som stötts på under projektet, där eventuella eller brister i studien erkänns.

6.1 Support Vector Regression

SVR är en kraftfull metod som öppnar stora möjligheter. Användingen är relativt enkelt men kan lösa komplexa icke-linjära problem. Metoden har fördelen att den kan lära sig samband mellan flertal indata till utdatan. Det finns dock nackdelar/-begränsningar.

För det första är metoden väldigt käsnlig för val av hyperparametrarna och fel parametrar kan ge dåligt resultat eller överpassning. Målet är att få en modell som är så generell som möjligt men med ett acceptabelt fel. Det blir lätt en avvägning där man får välja om man vill minimera felet men riskera överpassning eller det omvända. Att optimera kombinationen av parametrarna är svårt, tidskrävande och datorkrävande.

För det andra ger kan SVR ses som en "black box". Även om modellen presterar bra vet man inte vilka samband modeller tar fram mellan de indatan och utdatan. Metoden ger alltså användaren inga insikter om potentiella samband, speciellt om flera in-variabler används. Modellen kan desutom ge intryck av att samband finns mellan indatan och den förutspådda datan även om det egentligen inte finns.

Slutligen är metoden relativt beräknings-intensiv, speciellt för större datamängder. Det innebär att det kan ta lång tid för modellen att ta fram de förutspådda värdena.

6.2 Support Vector Regression och C_μ

Det blev tydligt utifrån resultatet och diskussion att då $C_\mu^{k-\omega}$ beräknats med hjälp av DNS data att flertalet värden var ofysikaliska.

Till följd av detta drar projektgruppen slutsatsen att optimering av $C_\mu^{k-\omega}$ inte är möjligt utifrån arbetets avgränsningar och därför avslutades det försöket. Fokus lades istället på den icke-linjära Eddy-viscosity modellen, kap. 2.1.2. Med detta menas

den modell som istället för $C_{\mu}^{k-\omega}$ använder sig av konstanterna c_{1-3}. Projektgruppen gick därmed vidare med optimering av denna modell.

6.3 k-Nearest Neighbors Regression

Med hänsyn till träningshastigheten, träffsäkerheten, samt förekomsten av buller, kan vi dra slutsatsen att för stora mänger data, och där buller inte är av stor vikt, kan kNN vara ett bra alternativ till mer beräkningsmässigt krävande regressionsmodeller. Dessutom har metoden inga begränsningar på dimensioner av in- och utdata, och kan därför lätt överföras från fall till fall.

6.4 Neuralt Nätverk

Slutsatsen som kan dras från arbetet samt diskussionen av det neurala nätverket är att modellen behöver tränas på betydligt mer data för att kunna användas för fler strömmningsfall. Det är nästintill förväntat att få de dåliga resultaten som syns i figur 4.11 och figur 4.12 då modellen endast tränats på 80% av ett enda strömmningsfall.

Projektgruppen anser däremot att en stor potential och förbättringsmöjlighet för området påvisats, speciellt för den ickelinjära *Eddy viscousity* modellen. Detta framgår främst i figur 4.9 och figur 4.10, där modellen förutspår Reynoldsspänningarna bättre än de vedertagna konstanterna från [6].

Detta resulterar slutligen i att med större datatillgång samt tidstillgång kan en förbättrad modell hittas med hjälp av metoden projektgruppen använt.

6.5 Slutgiltig bedömning

Slutsatsen som kan dras för arbetet i sin helhet är att det finns stor potential för att implementera ML och neurala nätverk inom strömmningsmekanik och turbulensmodellering. Trots att projektgruppen inte lyckats påvisa tydlig generell förbättring har en stor potential visats, inte minst för användningen av neurala nätverk i kombination med NEVM samt k-d Tree och kNN. Projektgruppen uppmanar därför till vidare forskning på dessa metoder och tillvägagångssätt.

Litteraturförteckning

[1] Frans T M Nieuwstadt et al. Turbulence Introduction to Theory and Applications of Turbulent Flows. Tech. rep.

[2] Sudarshana S Rao and Santosh R Desai. Machine learning based traffic light detection and ir sensor based proximity sensing for autonomous cars. In Proceedings of the International Conference on IoT Based Control Networks & Intelligent Systems – ICICNIS, 2021. URL http://dx.doi.org/10.2139/ssrn.3883931.

[3] Andreas Lindholm, Niklas Wahlstrom, Fredrik Lindsten, and Thomas Sch ¨ on. ¨ Machine Learning: A First Course for Engineers and Scientists. Cambridge University Press, 03 2022. ISBN 9781108843607. doi: 10.1017/9781108919371. URL http://smlbook.org/

[4] L. Davidsson, "Fluid mechanics, turbulent flow and turbulence modeling", opublicerad, March 18, 2023, URL: https://www.tfd.chalmers.se/~lada/postscript_files/solids-and-fluids_turbulent-flow_turbulence-modelling.pdf

[5] L. Davidsson, "An Introduction to Turbulence Models", opublicerad, August 1, 2022, URL: https://www.tfd.chalmers.se/~lada/postscript_files/kompendium_turb.pdf

[6] T. J. Craft, B. E. Launder, och K. Suga. Prediction of turbulent transitional phenomena with a nonlinear eddy-viscousity model. *International Journal of Heat and Fluid Flow*, Vol. 18, No. 1, February 1997.

[7] Basak, Debasish & Pal, Srimanta & Patranabis, Dipak. (2007). Support Vector Regression. Neural Information Processing – Letters and Reviews. 11.

[8] Shiyu Ji, Support vector regression (prediction) with different thresholds ε. As ε increases, the prediction becomes less sensitive to errors."https://en.wikipedia.org/wiki/Support_vector_machine#/media/File:Svr_epsilons_demo.svg, hämtad: 2023-04-20

[9] Samet, Hanan (2006). "Foundations of Multidimensional and Metric Data Structures". K-D Trees. pages: 50-63 https://books.google.se/books?id=vO-NRRKHG84C&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false, hämtad: 2023-05-09

[10] Wikimedia Commons, File:3dtree.png. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=589909 (visited on 2023-05-10)

[11] Frank Rosenblatt,The perceptron: a probabilistic model for information storage and organization in the brain.", Psychological review, 1958, Vol. 65-6, pages: 386-408

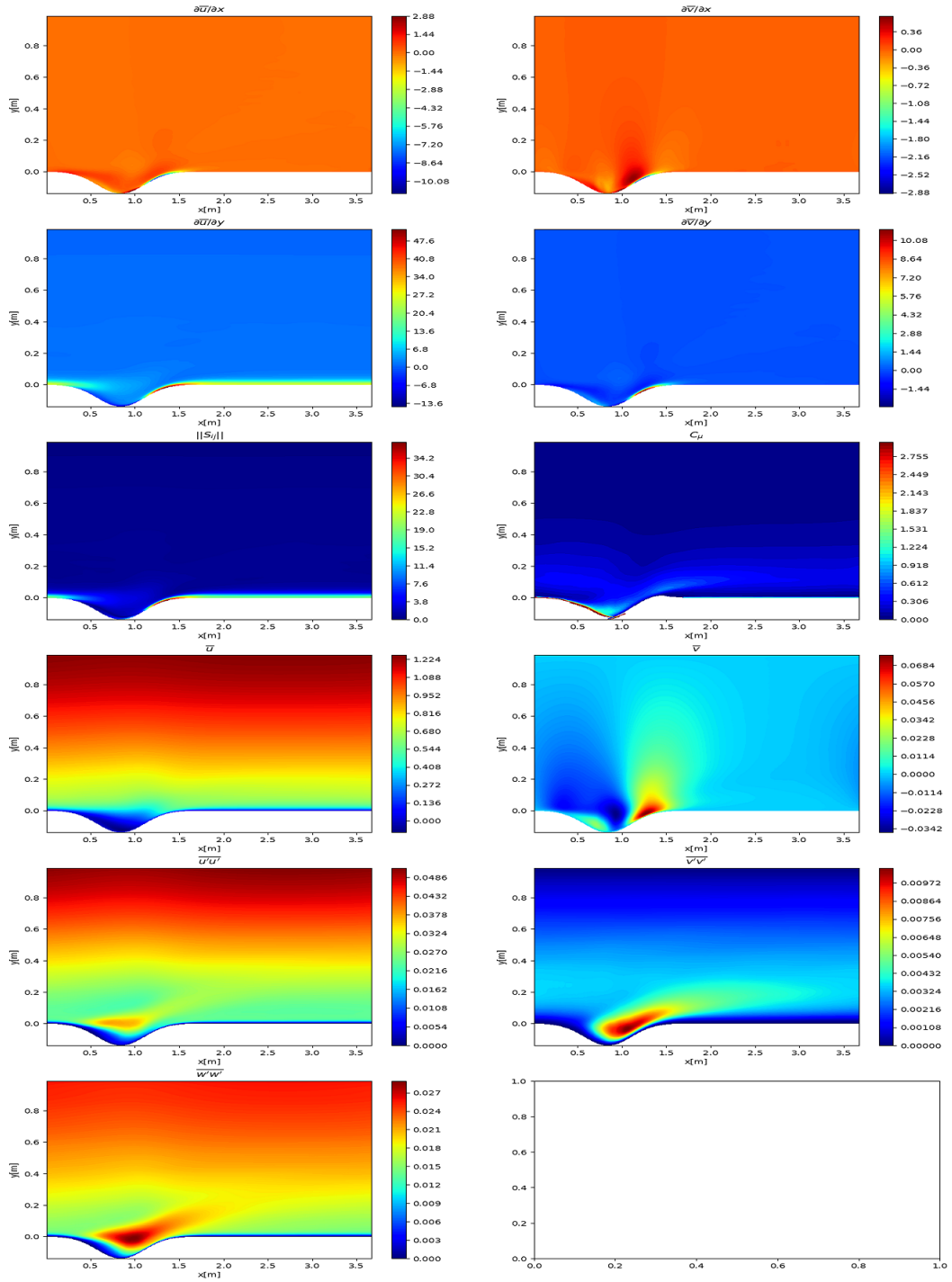
[12] Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron (2016). "6.5 Back-Propagation and Other Differentiation Algorithms". Deep Learning. MIT Press. pages: 200–220. https://www.deeplearningbook.org/contents/mlp.html#pf25

[13] Lars Davidsson, privat kommunikation, Jan. 2023.

[14] Myoungkyu Lee and Robert D. Moser, Direct numerical simulation of turbulent channel flow up to Re_tau = 5200,2015, Journal of Fluid Mechanics, vol. 774, pp. 395-415

A

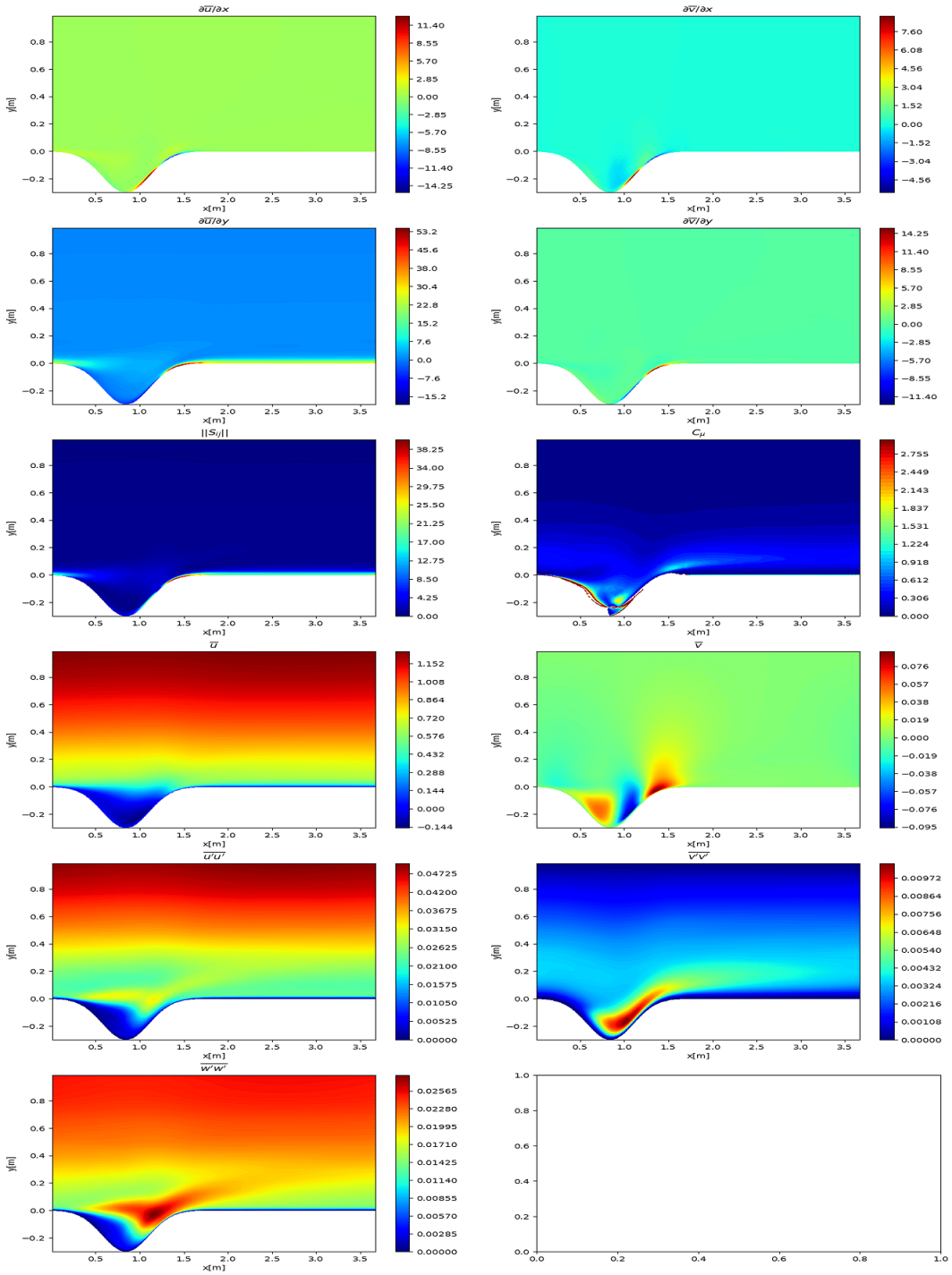
Data från small wave fallet



Figur A.1: DNS data illusterad med contur-graf över small-wave"

B

Data från large wave fallet



Figur B.1: DNS data illustrerad med contur-graf över Large-wave

C

Git Repository

Länk till Git repository

Institutionen för
Mekanik och Maritima vetenskaper
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sverige 2023
www.chalmers.se

CHALMERS