Radielle basisfunksjoner som adaptiv kollokasjonsmetode

Tomas Nicolajsen

1. juni 2014

Takk!

Takk til mine veiledere Tor Sørevik og Carina Bringedal. De har vært avgjørende for orienteringsevnen i et svært kupert matematisk landskap.

Takk til min kjære Benedikte for sin urokkelige tro på mine evner, selv i situasjoner hvor alt skulle tilsi det motsatte.

Tomas, mai 2014

Forord

Hensikten med oppgaven er å undersøke en adaptiv kollokasjonsmetode basert på radielle basisfunksjoner. Målet er å finne ut om man med enkle kriterier for senter- og formparameterfordelingen kan oppnå en fordeling som gjenspeiler egenskaper ved problemet, og om man med det kan oppnå en mer effektiv og stabil kollokasjonsmetode.

Innhold

Innledning

Motivasjon

Numerisk simulering av fysiske prosesser har i løpet av dens spede opprinnelse på 50-tallet vært igjennom en rivende utvikling. Utviklingen er todelt. På den ene siden har man opplevd en omtrent eksponensiell vekst hva angår datamaskinens regnekraft, noe som drastisk har forskjøvet grensene for hva som er mulig å oppnå av nøyaktighet i simuleringene.

På den annen side, noe som er av enda større betydning, er utviklingen av numerisk teknologi. Det vil si metoder som utnytter den stadig økende, men likevel begrensete, regnekraften mer effektivt til fordel for bedre numeriske simuleringer.

Viktig for effektiviteten og nøyaktigheten til kollokasjonsmetoder er fordelingen av kollokasjonspunktene i simuleringens domene. Ideelt sett burde fordelingen være avhengig av fysikken til problemet. Om man vil simulere en enkelt bølges forflytning vil det være fordelaktig med kollokasjonspunkt i det området hvor bølgen til enhver tid befinner seg.

Etablerte pseudospektralmetoder benytter seg av nodefordelinger som er bestemt av forhold i selve metoden, ikke fysikken til problemet metoden skal simulere.

Endelig differanse- og endelig elementmetoder har større frihet hva angår nodefordelingen, men metodene er fremdeles avhengig av en viss struktur på nodene seg imellom. Denne strukturen er krevende å opprettholde, både når det gjelder implementering og når det gjelder bruk av regnekraft.

Denne oppgaven tar for seg kollokasjonsmetoder basert på radielle basisfunksjoner. Radielle basisfunksjoner gir opphav til gitterfrie metoder, noe som i motsetning til spektralmetoder gjør dem svært fleksible ovenfor problemer med irregulær geometri i domenet. I motsetning til pseudospektralmetoder

må ikke kollokasjonspunktene settes sammen med tensorprodukt. Dette åpner for en mer effektiv nodefordeling i problemer med flere dimensjoner.

Radielle basisfunksjoner (RBF) er også kjent for å kunne gi opphav til metoder med eksponensielle konvergensrater. Dette gir håp om at man med radielle basisfunksjoner kan lage en metode som kombinerer egenskaper på tvers av etablerte teknologier - endelig elementmetoders fleksibilitet og spektralmetoders effektivitet.

Om oppgaven

Oppgaven undersøker en ikke-standard metode for numerisk simulering av partielle differensialligninger. Det finnes lite teoretisk fundament som kan understøtte resultatene fra simuleringene. Derfor forsøkes en eksperimentell tilnærming til den numeriske matematikken. Antakelsene som er utgangspunktet for den adaptive metoden kan sees på som hypoteser bygget på erfaringene fra forsøkene som gjøres i kapittel 2.

Den adaptive metoden blir testet på en rekke modellproblemer med og uten tidsavhengighet. Resultatene blir sammenlignet med den eksakte løsningen i tilfeller hvor den er tilgjengelig. I tillegg blir det sammenlignet med Chebyshevs pseudospektralmetode, standard RBF-metoden, endelig elementmoden samt sammenlignbare adaptive metoder i litteraturen.

Kapittel 1 beskriver grunnleggende konsepter innen numerisk approksimasjon av partielle differensialligninger. De påfølgende numeriske forsøkene blir beskrevet og tolket i lys av disse konseptene.

Kapittel 2 introduserer radielle basisfunksjoner og kollokasjonsmetoden som er grunnlag for den adaptive metoden. En rekke numeriske forsøk med senter- og formparameteren blir gjort. Hensikten med dette er å studere hvordan dette påvirker approksimasjonsfeilen og kondisjonstallet til kollokasjonsmatrisen.

Kapittel 3 beskriver den adaptive kollokasjonsmetoden.

Kapittel 4 presenterer og diskuterer resultater fra den adaptive kollokasjonsmetoden.

Kapittel 5 oppsummerer oppgaven, og gir konklusjoner på bakgrunn av de numeriske resultatene.

Kapittel 1

Simulering av partielle differensiallikninger

Den numeriske utfordringen med differensialligninger kan i hovedsak knyttes til hvordan en best mulig kan approksimere den deriverte av en ukjent funksjon. Utfordringen bunner i en fundamental forskjell mellom differensialmatematikken og datautregninger: førstnevnte har et uproblematisk forhold til infinitesimale størrelser, mens sistnevnte har det motsatte. Dette er en tøff, men ikke uoverkommelig barriere.

Overgangen fra kontinuerlige til diskrete variabler kan ved hjelp av konsepter fra numerisk approksimasjon gjøre en tilsynelatende uoverkommelig barriere mulig. Dette gjøres uten å gå på bekostning av matematikkens høye krav til presisjon og rigorøsitet.

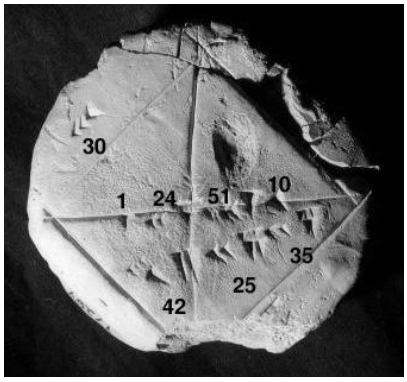
1.1 Numerisk approksimasjon og basisfunksjoner

Numerisk approksimasjon kan sies å være matematikkens livsnerve når det kommer til fagets betydning i det 'virkelige' livet. Numerisk approksimasjon utgjør en nødvendig overgang fra symboltung abstrakt matematikk til konkrete anvendbare tall.

En matematiker kan for eksempel langt på vei være tilfreds med å se på $\cos(\theta)$ som en projeksjon av en koordinat fra enhetssirkelen ned på x-aksen. For en tømrer, derimot, er dette lite hensiktsmessig kunnskap. I arbeidet med sitt

sperretak med helningsvinkel på tretti grader er han langt mer interessant i tallet cos(30). Den hjelpsomme matematikeren vil kanskje regne dette ut for ham og gi ham svaret $\cos(30) = \sqrt{3}/2$. Tømreren er fremdeles ikke fornøyd, $\sqrt{3}/2$ er ikke et tall som finnes på målbåndet.

For å kunne gi tømreren et anvendbart tall er numerisk approksimasjon av enten cos(30) eller $\sqrt{3}$ nødvendig. Numeriske approksimasjoner har røtter langt tilbake i matematikk. Av de tidligste matematiske skriftene som er funnet, er trolig en numerisk utregning av $\sqrt{2}$ en av de første. Slike utregninger har røtter i tidsepoker lenge før datamaskinen et foretrukket hjelpemiddel. Figur 1.1 viser approksimasjon av $\sqrt{2}$ gjort på en leirtavle fra ca. år 1800-1600 f.Kr.



Figur 1.1: *Approksimasjon av $\sqrt{2}$ uten datamaskin. Bilde fra Wikipedia.*

Konseptet om basisfunksjoner brukes utbredt i beskrivelsen av numeriske metoder. I tilfellet med approksimasjon av $cos(\theta)$ kan for eksempel dens Taylorserie sees på som en numerisk metode for å beregne brukbare tallverdier:

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - ... \qquad ,\forall x \tag{1.1}$$

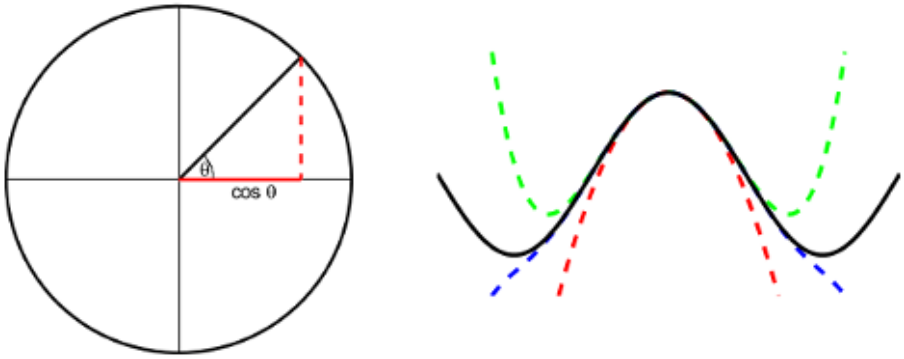
Basisfunksjonene til en Taylorserie omkring $x = 0$ blir dermed polynomer av økende grad $1, x, x^2, ...$ Koeffisientene til basisfunksjonene finnes med formelen $\frac{f^n(0)}{n!}$ hvor f er funksjonen som skal approksimeres, i dette tilfellet $\cos(\theta)$. Figur 1.2 illustrerer denne tilnærmingen gjort med Taylor basisfunksjoner.

Et fellestrekk ved de numeriske metodene som omtales i denne oppgaven er at de baseres på antagelsen om at den numeriske approksimasjonen kan representeres som en begrenset lineær kombinasjon av basisfunksjoner. Det vil si at den approksimerte løsningen u_N matematisk kan formuleres som

$$u \approx u_N = \sum_{j=1}^{N} \lambda_j \phi_j(x), \tag{1.2}$$

Her er λ_j koeffisientene til de N antall basisfunksjonene $\phi_j(x), j = 1...N$.

I motsetning til for taylorapproksimasjonen (1.1) finnes det på den generelle formen (1.2) ikke formeluttrykk for koeffisientverdiene. Hvilken fremgangs-



Figur 1.2: *Cosinus sett fra to perspektiv - som projeksjon (venstre) og som approksimant (høyre). Stiplete linjer er taylorserier med henholdsvis 2, 3 og 4 ledd*

måte som brukes for å finne koeffisientene er i mange tilfeller beskrivende for hvilken numerisk metode man har med å gjøre.

1.2 Kollokasjonsmetoder

Gitt at differensialligningen som skal approksimeres er på formen

$$Lu = f \tag{1.3}$$

Her er L en lineær differensialoperator, u den ukjente løsningen og f en kjent funksjon.

I utledningen av en numerisk metode følges basisfunksjonsstrategien som beskrevet over, og u erstattes med den approksimerte u_N fra (1.2):

$$Lu \approx Lu_N = L\sum_{j=0}^{N} \lambda_j \phi_j(x) = f(x) \tag{1.4}$$

Residualen denne substitusjonen gir defineres som [4]

$$E(x) = f(x) - Lu_N(x)$$

I tillegg til at valg av basisfunksjoner er avgjørende, er også måten approksimasjonsfeilen minimeres på avgjørende for den numeriske metoden. Utvilsomt vil man minimere approksimasjonsfeilen på en 'best mulig' måte.

Denne oppgaven tar for seg kollokasjonsmetoder. Med det menes at E settes lik null i et gitt antall punkt x_i:

$$E(x_i) = f(x_i) - Lu_N(x_i) = 0 \tag{1.5}$$

Lineærkombinasjonen av basisfunksjoner settes så inn for u_N

$$f(x_i) - Lu_N(x_i) = 0 \tag{1.6}$$

$$L\sum_{j=0}^{N} \lambda_j \phi_j(x_i) = f(x_i) \tag{1.7}$$

Dette er et lineært ligningssystem som kan skrives på matriseformen $\boldsymbol{A_L\lambda} = \boldsymbol{f}$:

$$\begin{pmatrix} L\phi_1(x_1) & L\phi_2(x_1) & \dots & L\phi_N(x_1) \\ L\phi_1(x_2) & L\phi_2(x_2) & \dots & L\phi_N(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ L\phi_1(x_N) & L\phi_2(x_N) & \dots & L\phi_N(x_N) \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_N \end{pmatrix} = \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_N) \end{pmatrix} \tag{1.8}$$

Merk at her er $E(x_i)$ satt til null i N antall punkt med den hensikt å oppnå en kvadratisk matrise som forhåpentligvis er inverterbar.

Spektralmetoder som er basert på denne fremgangsmåten kalles kollokasjonsmetoder. De har mye til felles med interpolasjon av funksjoner. Det eneste som faktisk skiller matrisen i likning (1.8) fra interpolasjonsmatrisen til funksjonen f er tilstedeværelsen av differensialoperatoren L.

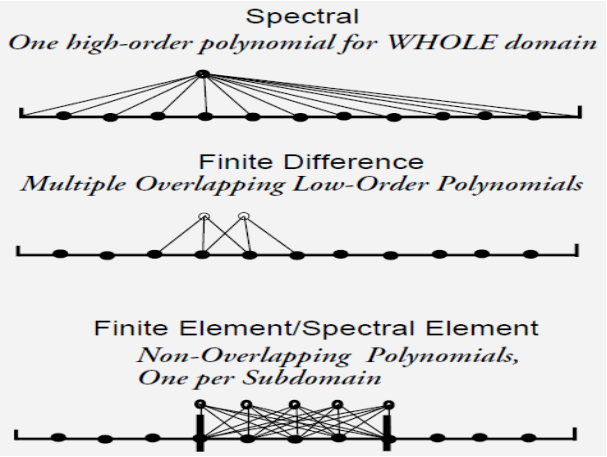
De tre store

Utviklingen av numeriske metoder for differensialligninger kan beskrives som dominert av de 'tre store' teknologiene som hver for seg oppstod i tiårene etter 1950-tallet [17]. Med det menes, i kronologisk rekkefølge, endelig differansemetoder, endelig elementmetoder og til slutt spektralmetoder.

I lys av basisfunksjonskonseptet kan man se at metodene har visse likhetstrekk, men også fundamentale forskjeller. De har alle til felles å være basert på polynomer som basisfunksjoner. I tillegg deler de egenskapen å være basert på en form for gitterinndeling av domenet. Det vil si at basisfunksjonene er avhengig av at kollokasjonspunktene er plassert i et innbyrdes mønster i forhold til hverandre.

Figur 1.3 sammenligner 'de tre store' med utgangspunkt i hvordan basisfunksjonene defineres.

I denne oppgaven fungerer Chebyshevs basisfunksjoner som representant for pseudospektralmetodene. Chebyshev er globale trigonometriske basisfunksjoner. Verdien i én node avhenger av alle de andre nodene. For å regne



Figur 1.3: *Illustrasjon av basisfunksjonene til 'de tre store'. Figuren er hentet fra Boyd [4]*

ut koeffisientverdiene må en full matrise inverteres. Om funksjonen som approksimeres er uendelig mange ganger kontinuerlig deriverbar gir denne type metoder eksponensiell konvergensrate [4].

Endelig differansemetoder baseres på laverere ordens polynomer med overlapp. Slike metoder er mer fleksible overfor geometri med irregulære render. De gir opphav til glisne matriser som kan inverteres langt mer effektivt enn fulle. Ulempen med slike basisfunksjoner er at de gir opphav til metoder med kun algebraisk konvergensrate.

Endelig elementmoder kan sees på som en videreutvikling av endelig differanse. Her brukes istedet polynomer uten overlapp. Det kan brukes høyere ordens polynomer på hvert subdomene som kan gi høyere konvergensrate enn endelig differanse. Triangulering av domener er ressurskrevende, og spesielt vanskelig blir det når dimensjonen på problemet øker.

1.3 Derivasjonsmatriser

Approksimasjonen (1.2) kan evalueres i punktene $x_1, ..., x_N$ ved matrise-vektor produktet $\boldsymbol{u_N} = \boldsymbol{A\lambda}$:

$$\begin{pmatrix} u(x_1) \\ u(x_2) \\ \vdots \\ u(x_N) \end{pmatrix} = \begin{pmatrix} \phi_1(x_1) & \phi_2(x_1) & \dots & \phi_N(x_1) \\ \phi_1(x_2) & \phi_1(x_2) & \dots & \phi_N(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_N) & \phi_2(x_N) & \dots & \phi_N(x_N) \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_N \end{pmatrix} \tag{1.9}$$

Matrisen her kalles evalueringsmatrisen, og er av dimensjon NxN.

Koeffisientvektoren $\boldsymbol{\lambda}$ finnes formelt ved å venstremultiplisere (1.9) med $\boldsymbol{A}^{-1}$

$$\boldsymbol{A}^{-1}\boldsymbol{u_N} = \boldsymbol{A}^{-1}\boldsymbol{A}\boldsymbol{\lambda}$$

som gir $\boldsymbol{\lambda} = \boldsymbol{A}^{-1}\boldsymbol{u_N}$. Ved å kombinere uttrykket med $\boldsymbol{A_L}\boldsymbol{\lambda} = \boldsymbol{f}$ fra (1.8) elimineres koeffisientvektoren og vi har følgende uttrykk:

$$\boldsymbol{A_L}\boldsymbol{A}^{-1}\boldsymbol{u_N} = \boldsymbol{f} \tag{1.10}$$

Her er det vanlig å sette $\boldsymbol{A_L}\boldsymbol{A^{-1}} = \boldsymbol{D}$, og vi har at

$$\boldsymbol{D}\boldsymbol{u}_N = \boldsymbol{f} \tag{1.11}$$

hvor $\boldsymbol{D}$ kalles derivasjonsmatrisen:

Uttrykket (1.11) er den diskrete analogien til det kontinuerlige differensialproblemet (1.3). Merk at uten randbetingelser er (1.3) ikke 'well-posed', det vil si at vi ikke har en entydig løsning. Dermed er det heller ikke å forvente at $\boldsymbol{D}$ er inverterbar når randetingelser ikke er implementert i matrisen.

1.4 Stabilitet og konvergens

De grunnleggende utfordringene når det kommer til approksimasjon av differensialproblemer kan knyttes til overgangen fra kontinuerlige til diskrete tall. Hensyn å ta når det gjelder numeriske simuleringen kan deles i flere kategorier.

Trunkeringsfeil

Dette er feil som oppstår ved å velge N antall basisfunksjoner i approksimasjonen. Trunkeringsfeilen defineres som

$$R_N = u - u_N$$

hvor u_N approksimerer u med N antall basisfunksjoner. I noen tilfeller kan man finne et uttrykk for størrelsen på trunkeringsfeilen. I tilfellet med taylorapproksimasjonen til $\cos(\theta)$ (1.1) er feilen gitt ved

$$R_2 = \frac{\sin(\xi)}{3!}x^3$$

hvor ξ er et tall mellom 0 og 1. Slike uttrykk er generelt ikke mulig å finne for (1.3).

Avrundingsfeil

En datamaskin er en binær maskin som gjør oppgaven med å representere alle reelle tall til en umulig oppgave. De fleste datamaskiner representerer tall så godt de kan ved hjelp av flyttallrepresentasjonen. Flyttall kan sees på som den numeriske analogien til den normaliserte standardformen (for eksempel $6,72 \cdot 10^9$).

Med flyttallskonvensjonen kan datamaskinen representere tall på formen

$$x = \pm mantissen \cdot 2^{eksponent}$$

Faktoren *mantissen* er et tall på binærform. Antallet siffer som tallet *mantissen* består av sier noe om hvor presist datamaskinen kan representere det reelle tallsystemet. MATLAB benytter seg av standarden 'IEEE dobbelpresisjon' i utregningene. Et 64 bit flyttall på denne formen har en mantisse bestående av 52 bit:

$$mantissen = \sum_{n=1}^{52} c_n 2^{-n} \tag{1.12}$$

hvor c_n er 0 eller 1 i binærsekvensen. Det siste leddet i rekkeutviklingen, $2^{-52} \approx 2 \cdot 10^{-16}$ kan sees på som den høyeste 'oppløsningen' når det kommer til hvor godt flyttallet kan gjengi et reelt tall. Det vil altså være 'hull' i den flyttallsbaserte tallinjen, men de vil være av størrelsesorden mindre enn $\mathcal{O}(10^{-16})$.

Kondisjonering og stabilitet

I analysen av numeriske beregninger skilles det mellom *problemets* kondisjonering og *algoritmens* stabilitet. Problemets kondisjonering har med problemets perturbative egenskaper å gjøre. Et velkondisjonert problem oppfører seg på en slik måte at en liten perturbasjon i input fører til en liten perturbasjon i output.

Etter diskretisering av $Lu = f$ hvor L er en operator, u og f er funksjoner, vil vi ha $\boldsymbol{D}\boldsymbol{u}_N = \boldsymbol{f}$ hvor $\boldsymbol{D}$ er en matrise, $\boldsymbol{u}_N$, $\boldsymbol{f}$ er vektorer. Dette er et lineært ligningssystem, og i denne oppgaven brukes matrisens kondisjonstall som et mål på metodens stabilitet.

Kondisjonstallet til matrisen, $\boldsymbol{A}$, defineres som [18]

$$\kappa(\boldsymbol{A}) = \|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\|,$$

her er $\|\cdot\|$ en vilkårlig matrisenorm, men i utregninger benyttes den induserte normen fra den euklidske vektornormen $\|\boldsymbol{x}\|_2 = \sqrt{x_1^2 + ...x_N^2}$. Det kan vises at matrisenormen dermed blir $\|\boldsymbol{A}\| = \sigma_{max}(\boldsymbol{A})$, den største singulærverdien til matrisen. Kondisjonstallet til matrisen, $\boldsymbol{A}$, beregnes da ved

$$\kappa(\boldsymbol{A}) = \sigma_{max}(\boldsymbol{A})/\sigma_{min}(\boldsymbol{A})$$

Når det gjelder vektornormer benyttes derimot maksimumnormen til fordel for den euklidske normen. Maksimumsnormen defineres som det elementet med den største tallverdien i vektoren, $\|\boldsymbol{x}\|_\infty = max(|x_1|, ..., |x_N|)$. I tilfeller hvor resultatene fra kollokasjonsproblemet blir sammenlignet med eksakte løsninger, blir sammenligningen gjort punktvis i kollokasjonspunktene. Dermed blir maksimumsnormen et naturlig valg etter tankegangen 'nøyaktigheten til metoden er ikke større enn det mest unøyaktige kollokasjonspunktet'.

For interpolasjonsproblemer blir resultatet sammenlignet i en punktmengde som *ikke* overlapper interpolasjonspunktene.

Som sagt skilles det mellom et problems kondisjonering, og algoritmens stabilitet. Dette er en skilnad som først og fremst eksisterer av teoretiske hensyn. Stabilitet er først og fremst et analyseverktøy for å bevise at det er mulig å lage en dataalgoritme som løser et kontinuerlig problem i den forstand at 'en stabil algoritme gir det nesten riktige svaret til det nesten riktige spørsmålet' [18].

Kjernen i kollokasjonsmetoder består i å finne koeffisientene til basisfunksjonene som ved å løse (1.11). Dette er et lineært ligningssystem på formen $\boldsymbol{Ax} = \boldsymbol{b}$, og den beregnede koeffisientvektoren $\tilde{\boldsymbol{x}}$ tilfredsstiller da

$$\frac{\|\tilde{\boldsymbol{x}} - \boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq \kappa(\boldsymbol{A})\epsilon_{maskin} \tag{1.13}$$

Maskinepsilon, ϵ_{maskin}, er det minste relative tallet som kan representeres i flyttalssystemet. Størrelsen er avhengig av mantissen. For et 64 bit flyttall, som beskrevet over, er $\epsilon_{maskin} = \mathcal{O}(10^{-16})$.

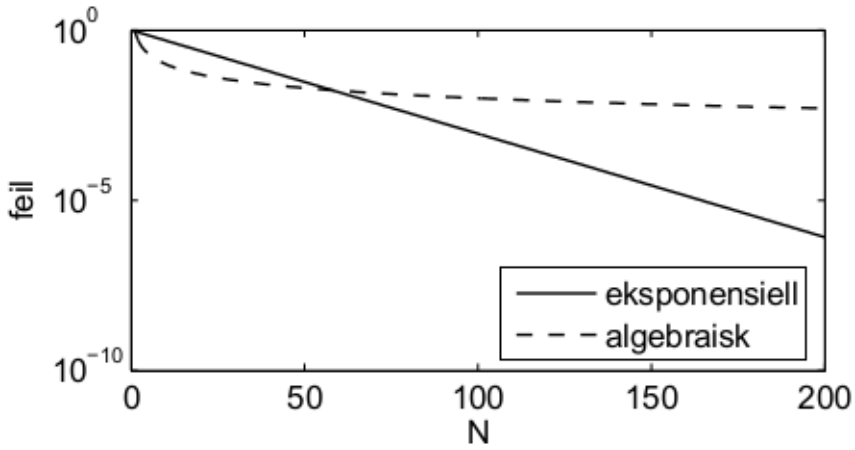
Uttrykket (1.13) gir opphav til tommelfingerregelen om at for et kondisjonstall 10^k kan man miste k siffer i presisjon som følge av matriseinverteringen, en feil som for kollokasjonsmetoder kommer på toppen i tillegg til feilkilder knytter til basisrepresentasjonen.

Konvergens og regneoperasjoner

Den største kostnaden i form av regnekraft når det kommer til kollokasjonsmetoder består av å regne ut koeffisientvektoren til basisfunksjonene. For mange tilfeller kan dette gjøres relativt rimelig. Pseudospektralmetoder fikk først sitt gjennombrudd etter det ble oppdaget at koeffisientene kunne regnes ut ved hjelp av en $\mathcal{O}(N \log N)$ 'Fast Fourier' metode.

For en generell full kollokasjonsmatrise uten noen spesiell struktur er inverteringen en kostbar $\mathcal{O}(N^3)$ operasjon. I begge tilfeller er det naturlig å måle konvergensraten som en funksjon av antall basisfunksjoner. For modellproblemer med eksakt løsning sammenlignes denne med den approksimerte løsningen med N antall basisfunksjoner i maksimumsnormen $\|\boldsymbol{u} - \boldsymbol{u_N}\|_\infty$.

Konvergensraten visualiseres ved å plotte $\|\boldsymbol{u} - \boldsymbol{u_N}\|_\infty$ som funksjon av N. Figur 1.4 illustrerer et tenkt tilfelle hvor den eksponensielle konvergensraten karakteriseres ved rette linjer i 'log-lin' plottet.



Figur 1.4: *Eksempel på plot av konvergensraten.*

1.5 Når tiden spiller inn – linjemetoden

Matematisk sett er det ingen forskjell mellom den tidsderiverte $\frac{\partial}{\partial t}$ og den deriverte med hensyn til den 'romlige koordinaten' $\frac{\partial}{\partial x}$. Likevel er det vanlig med forskjellige diskretiseringsstrategier i tid og rom.

Denne skilnaden mellom tid og rom oppstår i mange tilfeller naturlig om man tar hensyn til fysikken som modelleres. Høydeprofilen på en membran under

påvirkning av en kraft avhenger av hvordan den er spent fast på randen. Matematisk diskretiseres dette som et randverdiproblem. Deterministiske prosesser som skjer fremover i tid er alltid bestemt av tilstander bakover i tid. Matematisk diskretiseres dette som initialverdiproblem.

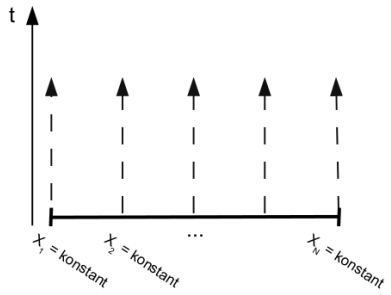
Varmeligningen $u_t = u_{xx}$ er et eksempel hvor både variabler for tid t og rom x inngår. For at denne ligningen skal modellere en fysisk diffusjonsprosess på en unik og entydig måte, må tilleggsbetingelser angis. I hvert tidsøyeblikk (t er konstant) kan varmeligningen sees på som et randverdiproblem. Fysisk sett vil det si at en må oppgi verdier for temperaturen i endepunktene. Det kan være konstante temperaturverdier (Dirichlet randbetingelser) eller en kan spesifisere visse randverdier for varmeflyten (Neumann randbetingelser).

Denne oppgaven fokuserer på romlig diskretisering med radielle basisfunksjoner. Tidsavhengige problemer representeres da på formen

$$\boldsymbol{u_t} = \boldsymbol{D}\boldsymbol{u_N} \tag{1.14}$$

hvor $\boldsymbol{D}$ er derivasjonsmatrisen. Uttrykket (1.14) er semidiskretisert, det vil si kun diskretisert i rom. Ligningen kan nå betraktes som et system av ordinære differensialligninger, nærmere bestemt et initialverdiproblem. Når en initialverdiene er gitt, kan systemet av differensialligningene approksimeres ved hjelp av en rekke løsningsmetoder som finnes for denne type problemer.

Strategien beskrevet over omtales som linjemetoden[1]. Varmeligningen reduseres til et system av ordinære differensialligninger ved å diskretiske den romlige koordinaten for deretter å integrere langs linjene hvor den romlige koordinaten holdes konstant.



Figur 1.5: *Illustrasjon av linjemetoden.*

[1]Method of lines

Stabilitet for linjemetoden

Hva angår stabiliteten for linjemetoder, er steglengden Δt avgjørende. I boken 'Spectral Methods in Matlab' kommer Trefethen [17] med følgende tommelfingerregel:

> Rule of Thumb.
> *The Method of Lines is stable if the eigenvalues of the (linearized) spatial discretization operator, scaled by Δt, lie in the stability region of the time-discretization operator.*

Anta vi vil bruke Eulers eksplisitte metode[13] for å approksimere den ordinære differensialligningen (1.14)

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + \Delta t \boldsymbol{D}\boldsymbol{u}_n = (\boldsymbol{I} + \Delta t \boldsymbol{D})\boldsymbol{u}_n \tag{1.15}$$

Her kan $\boldsymbol{u}_n$ uttrykkes som en lineær kombinasjon av egenvektorene ($\boldsymbol{v}_1, ..., \boldsymbol{v}_d$) med koeffisientene ($c_1^n, ..., c_d^n$) til $\boldsymbol{D}$: $\boldsymbol{u}_n = \sum_{i=1}^{d} c_i^n \boldsymbol{v_i}$

$$\boldsymbol{u}_{n+1} = (\boldsymbol{I} + \Delta t \boldsymbol{D})\boldsymbol{u}_n = (\boldsymbol{I} + \Delta t \boldsymbol{D}) \sum_{i=1}^{d} c_i^n \boldsymbol{v_i} = \sum_{i=1}^{d} c_i^n (\boldsymbol{v_i} + \Delta t \boldsymbol{D}\boldsymbol{v_i}) \tag{1.16}$$

Siden $\boldsymbol{v_i}$ er en egenvektor til $\boldsymbol{D}$, har vi at $\boldsymbol{D}\boldsymbol{v_i} = \lambda_i \boldsymbol{v_i}$ hvor λ_i er en egenverdi til $\boldsymbol{D}$. Dermed blir:

$$\boldsymbol{u_{n+1}} = \sum_{i=1}^{d} c_i^n (\boldsymbol{v_i} + \Delta t \boldsymbol{D}\boldsymbol{v_i}) = \sum_{i=1}^{d} c_i^n (\boldsymbol{v_i} + \Delta t \lambda_i \boldsymbol{v_i}) = (1 + \Delta t \lambda_i) \sum_{i=1}^{d} c_i^n \boldsymbol{v_i} \tag{1.17}$$

I tillegg vet vi at $\boldsymbol{u_{n+1}}$ kan uttrykkes som en lineær kombinasjon av egenvektorer

$$\boldsymbol{u_{n+1}} = \sum_{i=1}^{d} c_i^{n+1} \boldsymbol{v_i} \tag{1.18}$$

Sammenligner vi koeffisientene til de to uttrykkene for $\boldsymbol{u_{n+1}}$ har vi at

$$c^{n+1} = (1 + \Delta t \lambda_i) c^n \tag{1.19}$$

For at iterasjonen skal konvergere må koeffisientene være bundet, det vil si at et sted i iterasjonen må $c^{n+1} \leq c^n$, det vil si

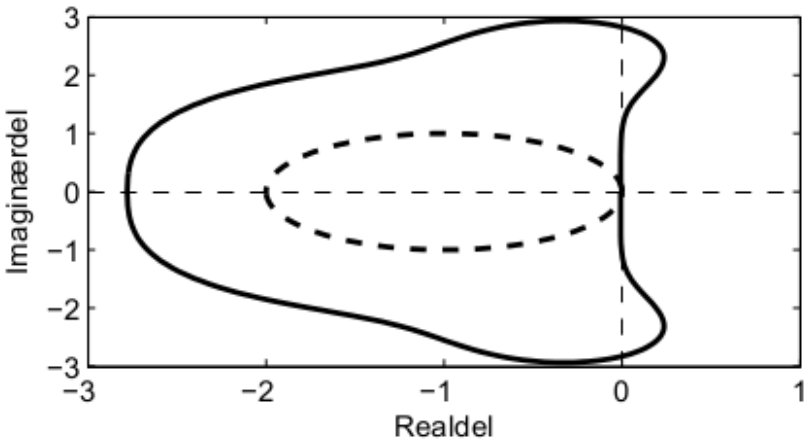
$$|1 + \Delta t \lambda| < 1 \tag{1.20}$$

Dette er uttrykket for stabilitetsregionen til den eksplisitte eulermetoden. Geometrisk sett, siden egenverdiene derivasjonsmatrisen ofte er komplekse

tall, vil det si at de skalerte egenverdiene må ligge i en sirkel med senter i $z = -1$ og radius lik én for at den eksplisitte eulermetoden skal være stabil.

Den lokale trunkeringsfeilen til metoden er av orden $\mathcal{O}((\Delta t)^2)$. Dette er feilen man kan forvente for hvert tidssteg. Siden antall tidssteg er av orden $\mathcal{O}(1/\Delta t)$ ender man opp med en global trunkeringsfeil av orden $\mathcal{O}(\Delta t)$.

Figur 1.6 viser stabilitetsregionen til den eksplisitte eulermetoden sammenlignet med den eksplisitte Runge Kutta 4.ordensmetoden [17].



Figur 1.6: *Stabilitetsregionen til eksplisitt Euler (stiplete linjer) og Runge Kutta 4. ordens eksplisitt metode (heltrukken linje).*

Kapittel 2

Radielle basisfunksjoner

Det som i dag kalles radielle basisfunksjoner har sin opprinnelse innen fagfeltet kartografi [7]. Der stod man ovenfor problemet med å beregne kontinuerlige høydekurver med utgangspunkt i gitte høydemålinger. Høydekurvene måtte være eksakt i målingspunktene, og de måtte oppføre seg på en 'naturlig' måte mellom målepunktene.

Radielle basisfunksjoner ble altså opprinnelig utviklet i forbindelse med interpolasjon av vilkårlig spredte noder i to dimensjoner. Den første anvendelsen kom trolig tidlig på slutten av sekstitallet med karttegningsprogrammet SYMAP [16].

Det var først på tidlig nittitalet radielle basisfunksjoner ble forsøkt brukt som kollokasjonsmetode. Kansa publiserte i 1990 'A scattered data approximation scheme with applications to computational fluid-dynamics' [12], kollokasjonsmetoden som i dag omtales som 'Kansas metode'.

De siste ti årene har det skjedd en oppblomstring innen forskjellige anvendelser på partielle differentialligninger. Forskjellige forslag på adaptive metoder med radielle basisfunksjoner har blitt publisert [14, 15, 6, 10].

Radielle basisfunksjoner (RBF) defineres som basisfunksjoner som er radielt symmetriske om et gitt senterpunkt. Da vil $\phi(r)$ være en radiell basisfunksjon med $r = \|\boldsymbol{x}-\boldsymbol{\xi}\|$ hvor $\boldsymbol{x}$ er funksjonens evalueringspunkt og $\boldsymbol{\xi}$ er et gitt senter.

Tabell (2.1) viser et utvalg mye brukte radielle basisfunksjoner. Det finnes langt flere muligheter alt etter om en ønsker funksjoner med lokalt eller globalt support, og med varierende grad av glatthet.

Alle simuleringene i denne oppgaven er gjort med multikvadratiske basisfunk-

Tabell 2.1: *Et utvalg globale radielle basisfunksjoner*

RBF	Definisjon ($r = \|\boldsymbol{x}\|$)
Multikvadratisk (MQ)	$\phi(r) = \sqrt{(\epsilon r)^2 + 1}$
Invers multikvadratisk (IMQ)	$\phi(r) = 1/\sqrt{(\epsilon r)^2 + 1}$
Gaussisk (GA)	$\phi(r) = e^{-(\epsilon r)^2}$
Kvadratisk Matèrn (QM)	$\phi(r) = e^{-\epsilon r}(3 + 3\epsilon r + (\epsilon r)^2)$

sjoner (appendiks A). Begrunnelsen for dette er at det er den mest brukte basisfunksjonen i litteraturen som beskriver RBF kollokasjonsmetoder. Dermed velges den for å gi adaptiv RBF metoden som beskrives senere et naturlig sammenligningsgrunnlag.

Basisfunksjonene i tabellen over har det felles å ha et globalt support (definert i hele domenet). I tillegg er de positiv definit, en egenskap som brukes til å bevise at koeffisientmatrisen som oppstår ved *interpolasjon* med disse funksjonene er inverterbar for en vilkårlig senterfordeling [7].

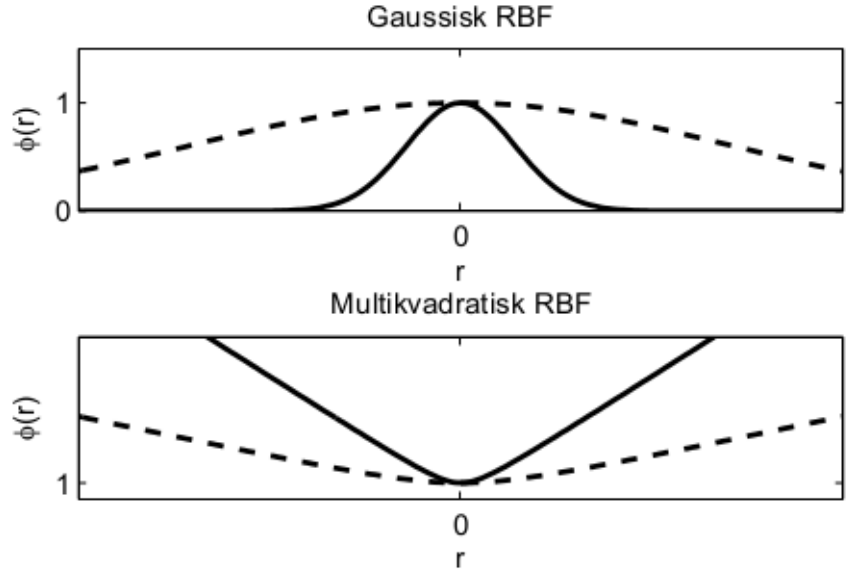
Formparameteren ϵ spiller en sentral rolle i RBF approksimasjon. Figur 2.1 illustrerer hvordan ϵ spiller inn på basisfunksjonens form. Ved økende ϵ-verdi vil generelt basisfunksjonens form bli brattere omkring senteret, og med senteret som funksjonens topp- eller bunnpunkt. Ved minkende ϵ-verdi vil funksjonen flates ut og bli stadig mindre lokalisert i domenet.

Formparameteren spiller en avgjørende rolle i nøyaktigheten til approksimasjonen med radielle basisfunksjoner, og mye oppmerksomhet har blitt gitt denne parameteren for å kunne forutsi dens effekt på approksimeringsresultatet.

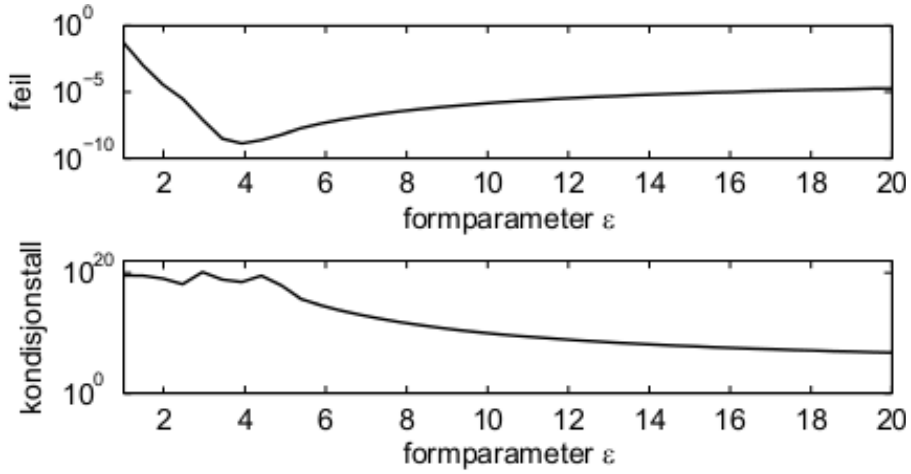
Numeriske eksperimenter viser at nøyaktigheten har en tendens til å øke i takt med lavere ϵ-verdi ('flatere' basisfunksjoner). Samtidig viser det seg at kondisjonstallet til koeffisientmatrisen øker med flatere basisfunksjoner. Trenden fortsetter til kondisjonstallet er så høyt at avrundingsfeil vil dominere og gjøre metoden ustabil for ϵ-verdier under dette.

Figur 2.2 illustrerer dette fenomenet ved interpolasjon av Runges funksjon $u(x) = 1/(25x^2 + 1)$ på intervallet $[-1, 1]$ med $N = 100$ basisfunksjoner.

Samspillet mellom nøyaktighet og kondisjonering ved approksimasjon med radielle basisfunksjoner kalles i litteraturen avveiningsprinsippet [7]. Det var lenge trodd at sammenhengen mellom kondisjonstallet og nøyaktigheten var et grunnleggende fenomen med radielle basisfunksjoner, og mye forskning er gjort på å finne optimale ϵ-verdier avhengig av hvilken radiell basisfunksjon



Figur 2.1: *Visualisering av to typer radielle basisfunksjoner i 1D (i 2D dreies de om senteraksen). Stiplete linjer viser 'lav' ϵ-verdi. Heltrukne linjer viser 'høy' ϵ-verdi.*



Figur 2.2: *Plot av feil og kondisjonstall for interpolasjon av runges funksjon* $u(x) = 1/(25x^2 + 1)$.

det er snakk om.

Oppfattelsen av avveiningsprinsippet som urokkelig har blitt endret med oppdagelsen av en alternativ algoritme for å finne koeffisientverdiene til basisfunksjonene [9]. Det ble vist at stabilitetsproblemene med lav ϵ -verdi først

og fremst skyldes forhold ved måten det lineære ligningssystemet settes opp på, og ikke kondisjoneringen til problemet.

Det bør merkes at oppdagelsen gjelder interpolajon, selv om det kan ventes at det trolig også finnes stabile algoritmer for kollokasjonsproblemer. Denne oppgaven bruker den klassiske måten å sette opp ligningssystemet, som i litteraturen beskrives som 'Kansas approach' eller 'RBF-Direct'.

RBF versus de Tre Store

Spektralmetoder, endelig differanse- og endelig elementmetoder har alle til felles av å være avhengig av et diskretisert domene hvor nodene må stå i et gitt forhold til hverandre. Man sier at nodene utgjør et gitter. Strukturen på gitteret er bestemt av den numeriske metoden, og ikke av egenskapene til problemet.

Radielle basisfunksjoner står i kontrast til dette ved å utmerke seg som en gitterfri metode. I prinsippet behøves ingen annen informasjon enn nodenes koordinater i domenet for å implementere metoden. Kompleksiteten til metoden er dermed uavhengig av geometrien og dimensjonen på domenet.

Håpet er at man med radielle basisfunksjoner kan si ja takk til begge deler – spektralmetodenes gode konvergensegenskaper og endelig differanse/element metoders håndtering av komplisert geometri. For å få dette til trengs det en måte å forvalte den nodemessige friheten en gitterfri metode medfører.

Tabell 2.2 oppsummerer de vesentligste egenskapene ved metodene.

Tabell 2.2: ***RBF sammenlignet med de Tre Store***

Metode	*Basis*	*Dekning*	*Overlapp*	*Gitterfri*	*Konvergens*
FD	Polynom	Lokal	Ja	Nei	Algebraisk
FEM	Polynom	Lokal	Nei	Nei	Algebraisk
Spektral	Polynom	Global	Ja	Nei	Eksponensiell
RBF	Radiell	Begge	Ja	JA!	Varierende

2.1 Kansas kollokasjonsmetode

Kansas kollokasjonsmetode (også kalt 'RBF-Direct') er navnet på en av de tidligste foreslåtte kollokasjonsmetodene med radielle basisfunksjoner. Det er

denne tilnærmingen som brukes for å sette opp det lineære ligningssystemet for kollokasjonsproblemene i denne oppgaven.

For et lineært differentialproblem på formen

$$Lu(\boldsymbol{x}) = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega \tag{2.1}$$

med Dirichlet randkrav

$$u(\boldsymbol{x}) = g(\boldsymbol{x}), \quad \boldsymbol{x} \in \partial\Omega \tag{2.2}$$

blir det korresponderende lineære ligningssystemet $\boldsymbol{K\lambda} = \boldsymbol{b}$ med Kansas tilnærming seende ut som følger:

$$\boldsymbol{K} = \begin{pmatrix} L\phi_{\epsilon_1}(\|\boldsymbol{x_1} - \boldsymbol{x_1}\|) & L\phi_{\epsilon_2}(\|\boldsymbol{x_1} - \boldsymbol{x_2}\|) & \dots & L\phi_{\epsilon_N}\|\boldsymbol{x_1} - \boldsymbol{x_N}\|) \\ \vdots & \vdots & & \vdots \\ L\phi_{\epsilon_1}(\|\boldsymbol{x_k} - \boldsymbol{x_1}\|) & L\phi_{\epsilon_2}(\|\boldsymbol{x_k} - \boldsymbol{x_2}\|) & \dots & L\phi_{\epsilon_N}(\|\boldsymbol{x_k} - \boldsymbol{x_N}\|) \\ \phi_{\epsilon_1}(\|\boldsymbol{x_{k+1}} - \boldsymbol{x_1}\|) & \phi_{\epsilon_2}(\|\boldsymbol{x_{k+1}} - \boldsymbol{x_2}\|) & \dots & \phi_{\epsilon_N}(\|\boldsymbol{x_{k+1}} - \boldsymbol{x_N}\|) \\ \vdots & \vdots & & \vdots \\ \phi_{\epsilon_1}(\|\boldsymbol{x_N} - \boldsymbol{x_1}\|) & \phi_{\epsilon_2}(\|\boldsymbol{x_N} - \boldsymbol{x_2}\|) & \dots & \phi_{\epsilon_1}(\|\boldsymbol{x_N} - \boldsymbol{x_N}\|) \end{pmatrix} \tag{2.3}$$

$$\boldsymbol{\lambda} = \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_k \\ \lambda_{k+1} \\ \vdots \\ \lambda_N \end{pmatrix}, \boldsymbol{b} = \begin{pmatrix} f(\boldsymbol{x_1}) \\ \vdots \\ f(\boldsymbol{x_k}) \\ g(\boldsymbol{x_{k+1}}) \\ \vdots \\ g(\boldsymbol{x_N}) \end{pmatrix} \tag{2.4}$$

Her er $\boldsymbol{x_1}, ..., \boldsymbol{x_k}$ sentre definert i domenet, mens $\boldsymbol{x}_{k+1}, ..., \boldsymbol{x}_N$ er sentre på randen av domenet. Merk at punktene $\boldsymbol{x_1}, ..., \boldsymbol{x_N}$ fungerer både som basisfunksjonenes senterpunkt og kollokasjonspunkt. Disse behøver ikke være samsvarende, men for enkelhets skyld er det vanlig å sette de til samme punktmengde. Det velges en individuell eller konstant formparameter. Individuell formparameter ϵ_i settes for hver enkelt basisfunksjon. Konstant formparameter vil si at alle basisfunksjonene har identisk ϵ-verdi ($\epsilon_i = \epsilon, \forall i$).

Matrisen $\boldsymbol{K}$ inneholder rader fra både matrisene $\boldsymbol{A_L}$ (1.8) og $\boldsymbol{A}$ (1.9). Matrisen blir brukt i den adaptive kollokasjonsmetoden i tilfeller hvor det trengs å regne ut den approksimerte løsningen i forskjellige punktmengder. Når koeffisientvektoren $\boldsymbol{\lambda}$ er funnet, kan den approksimerte løsningen evalueres i de vilkårlige evalueringspunktene $\boldsymbol{y}_1, ..., \boldsymbol{y}_M$ ved å venstremultiplisere koeffi-

sientvektoren med evalueringsmatrisen

$$\boldsymbol{E} = \begin{pmatrix} \phi_{\epsilon_1}(\|\boldsymbol{y_1} - \boldsymbol{x_1}\|) & \phi_{\epsilon_2}(\|\boldsymbol{y_1} - \boldsymbol{x_2}\|) & \dots & \phi_{\epsilon_N}\|\boldsymbol{y_1} - \boldsymbol{x_N}\|) \\ \vdots & \vdots & & \vdots \\ \phi_{\epsilon_1}(\|\boldsymbol{y}_M - \boldsymbol{x_1}\|) & \phi_{\epsilon_2}(\|\boldsymbol{y}_M - \boldsymbol{x_2}\|) & \dots & \phi_{\epsilon_N}(\|\boldsymbol{y}_M - \boldsymbol{x_N}\|) \end{pmatrix} \tag{2.5}$$

Kansas metode som beskrevet over er en generell beskrivelse av kollokasjon med radielle basisfunksjoner. Den gjøres først og fremst mer konkret ved å definere hvilken type basisfunksjon som benyttes. Tabellen 2.1 gir noen eksempler.

Lite analyse av denne metoden er gjort, og for en generell analyse med vilkårlig L og vilkårlig senter- og formparameterfordeling er dette en vanskelig oppgave. Det er ikke gjort funn i litteraturen som tar for seg nødvendige eller tilstrekkelige krav som kan garantere at matrisen $\boldsymbol{K}$ er inverterbar hverken for L, senter- eller formparameterfordelingen .

Det er dog kjent at $\boldsymbol{K}$ kan være singulær for visse tilfeller med konstant formparameter [7]. I tilfellet med individuell formparameter som benyttes i den adaptive metoden er inverterbarheten til $\boldsymbol{K}$ ikke diskutert i litteraturen.

Til tross for manglende analytiske resultater om viktige spørsmål, er Kansas tilnærming den vanligste tilnærmingen til kollokasjonsmetoder med radielle basisfunksjoner.

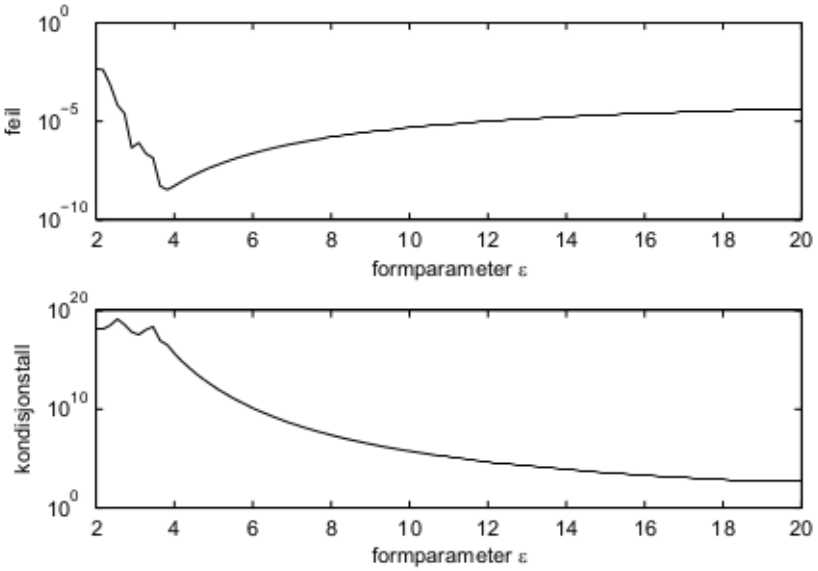
2.2 Numeriske forsøk med kollokasjonsmetoden

Konstant formparameter ϵ

For å sette opp kollokasjonsmetoden med Kansas tilnærming (2.3) må formparametre og en senterfordeling velges. En enklest mulig fordeling av sentre er uniform fordeling i domenet. Tilsvarende enkelt blir det å velge en konstant formparameter, det vil si samme verdi for alle basisfunksjonene.

Modellproblemet som brukes er poissonligningen (heretter kalt Poisson 1D), $u_{xx} = f$ på domenet $[-1, 1]$ hvor f finnes ved å ta utgangspunkt i den eksakte løsningen $u = 1/(25x^2 + 1)$. Dirichlet randkrav settes med utgangspunkt i den eksakte løsningen.

Figur (2.3) viser hvordan feilen og kondisjonstallet endres ved å variere den konstante formparameteren ϵ ved approksimasjon av poisson 1D. Antall sentre holdes konstant med uniform fordeling $x_k = -1 + 2(k-1)/99, \quad k = 1, ..., 100$. Tendensen (avveiningsprinsippet) er samme som for interpolasjon med RBF. I dette tilfellet ser man at feilen er lavest når formparameteren er like under $\epsilon = 4$.



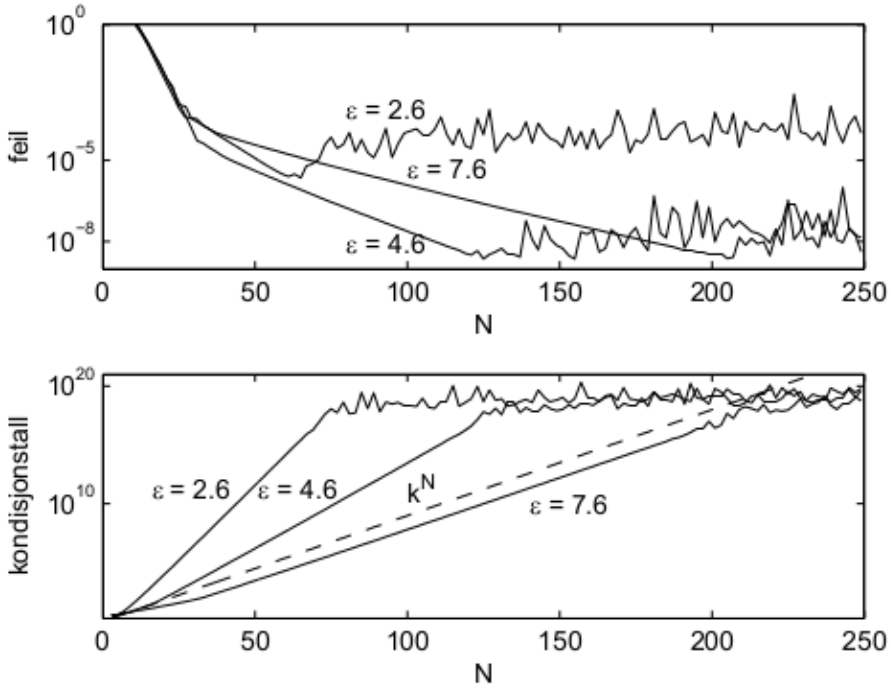
Figur 2.3: *Plot av feil og kondisjonstall som funksjon av forskjellige konstante ϵ-verdier for Poisson 1D. Uniform nodefordeling.*

Figuren er typisk for kollokasjon og interpolasjon hvor radielle basisfunksjoner med globalt support benyttes. Nøyaktigheten øker ved lavere ϵ-verdi. Visuelt vil dette si 'flatere' basisfunksjoner som er mindre lokalisert i domenet i motsetning til basisfunksjoner med høyere ϵ-verdi (se figur 2.1).

Approksimasjonsfeilen synker samtidig med at kondisjonstallet øker. Dette fortsetter helt til et nivå hvor avrundingsfeilen vil være av høyere orden enn trunkeringsfeilen, og prosessen med å regne ut koeffisientverdiene blir ustabil.

Figur 2.4 viser konvergensraten og kondisjonstallene for Poisson 1D for tre forskjellige formparametre. Når antall basisfunksjoner, N, holdes lavt gir alle tre formparametrene lik eksponensiell konvergensrate, men dette endres når N passerer omtrent 25 som figuren viser.

For praktiske problemer uten eksakt løsning er den optimale formparameteren ikke tilgjengelig. Figur 2.4 viser dessuten at den optimale formparamete-

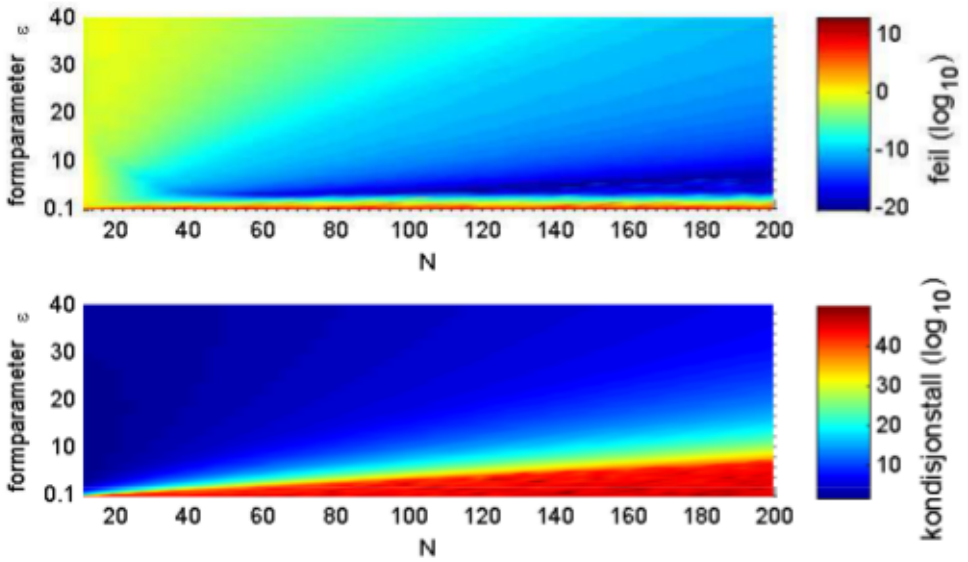


Figur 2.4: *Kollokasjon av Poisson 1D med konstant formparameter og uniform senterfordeling.*

ren befinner seg i et område hvor kondisjonstallet er så høyt at det i praksis er utilgjengelig for numeriske simuleringer. Boyd [5] beskriver dette området som '...jumping off a cliff; one may land softly, or one may be very sorry'. For poissonproblemer, hvor verdiene i $\boldsymbol{f}$ kan være samplet med et måleinstrument, vil en ørliten endring i føre til en endring i koeffisientverdiene med mange størrelsesordener.

Det numerisk utilgjengelige området blir større for økende antall sentre som figur 2.5 viser. Figuren viser '$\epsilon - N$ landskapet' til poisson 1D hvor approksimasjonsfeilen og kondisjonstallet er regnet ut for et tensorprodukt av forskjellige ϵ og N verdier.

Visuelt kan figuren tolkes som at kondisjonstallet øker etter hvert som de relative avstanden sentrene seg imellom blir mindre for økende N. Basisfunksjonene vil da ha økende overlapp, noe som vil si at de blir stadig vanskeligere å skille fra hverandre. Sett i forhold til hverandre blir de mer parallelle. Sagt med lineær algebra vil kolonnevektorene som utgjør løsningsrommet bli lineært avhengig når $\epsilon \to 0$.



Figur 2.5: *Approksimasjonsfeil med kondisjonstall til poisson 1D med varierende formparameter og økende antall sentre*

Konstant formparameter relativ til senteravstanden (uniform senterfordeling)

Figur 2.4 til de tre tilfellene av forskjellig formparameter ϵ viser at det er ikke kun konvergensraten som er eksponensiell. Kollokasjonsmatrisens kondisjonstall øker også eksponensielt.

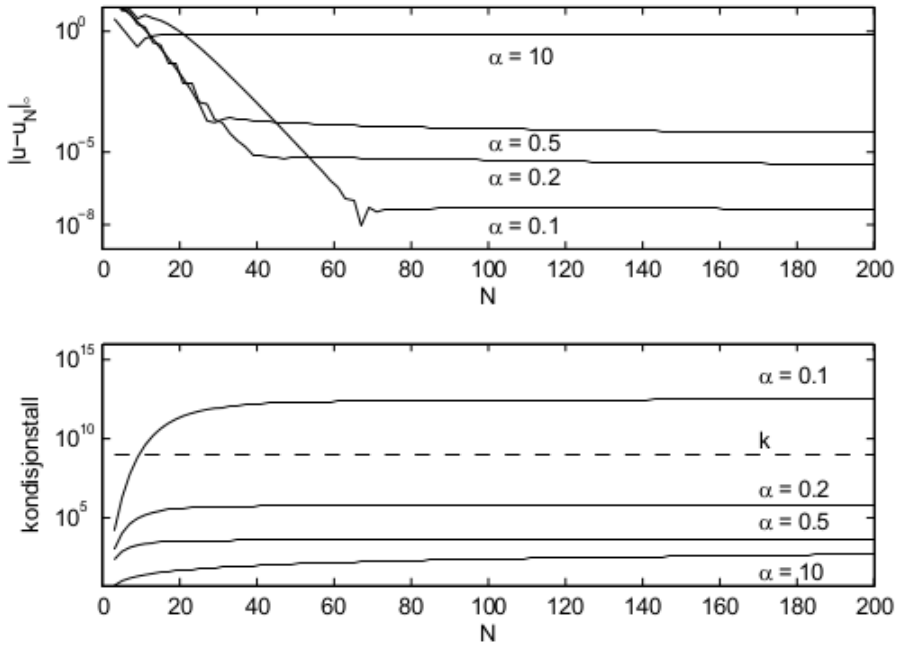
I en adaptiv metode hvor nye sentre legges til eller eksisterende sentre fjernes må formparametrene endres som funksjon av antall sentre. Hvis ikke er det lett å havne i et område i '$\epsilon - N$ landskapet' hvor kondisjonstallet til kollokasjonsmatrisen vil være ødeleggende for stabiliteten til metoden.

Å finne gode formparametre i et todimensjonalt parameterrom er vanskelig. På bakgrunn av tanken om at formparameteren må økes for å unngå dårlig kondisjonering når avstanden mellom sentrene økes, defineres formparameteren relativt til senteravstanden [5]. I tilfellet med uniformt fordelte sentre på domenet $[-1, 1]$ vil det si $\epsilon = \alpha \frac{2}{(N-1)}$ hvor N er antall sentre og α er en konstant.

Figur 2.6 viser hvordan approksimasjonsfeilen og kondisjonstallet nå forholder seg i forhold til antall sentre. Figuren viser at ved å definere formparameteren på denne måten kan en oppnå kondisjonstall som er uavhengig av antall sentre. Kontrasten til figur 2.4 er stor, hvor kondisjonstallet øker

eksponensielt med antall sentre.

Å definere formparameteren relativt til senteravstanden kan sees på som første steg i retning en adaptiv metode. Formparameteren blir da bestemt på en grov initiell senterfordeling, og blir senere endret relativt til nærmeste senter i prosessen.



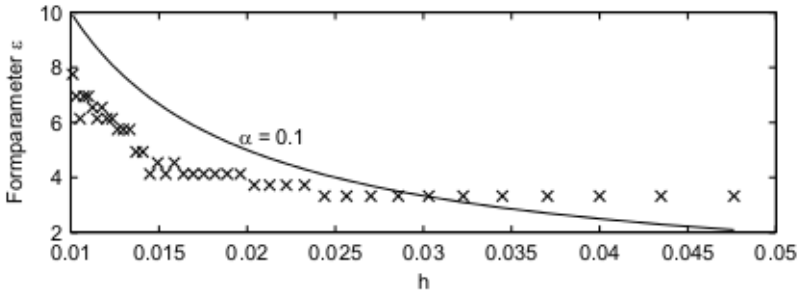
Figur 2.6: *Kollokasjon av Poisson 1D med konstant formparameter definert relativt til nærmeste senter. Uniform senterfordeling*

.

Figur 2.7 sammenligner denne strategien med de optimale formparametrene. De optimale formparametrene er funnet 'brute force' ved å søke igjennom et stort intevall med formparametre for hver N. Kondisjonstallene for de optimale ϵ er alle av orden større enn $\mathcal{O}(16)$. Figuren viser at det å velge formparametrene relativt til senteravstanden fanger opp trenden med økende optimal ϵ for økende N.

Individuell formparameter relativ til senteravstanden (ikke-uniform senterfordeling)

Forsøkene gjort hittil er alle basert på en uniform senterfordeling. I en adaptiv metode vil sentertettheten være varierende avhengig av forhold ved proble-



Figur 2.7: *Kryssene viser optimale formparametre ϵ. Linjen viser hvilke formparametre som oppnås ved å velge $\epsilon = \alpha h$ hvor $h = \frac{2}{N-1}$. Poisson 1D. Uniform senterfordeling på* $[-1, 1]$.

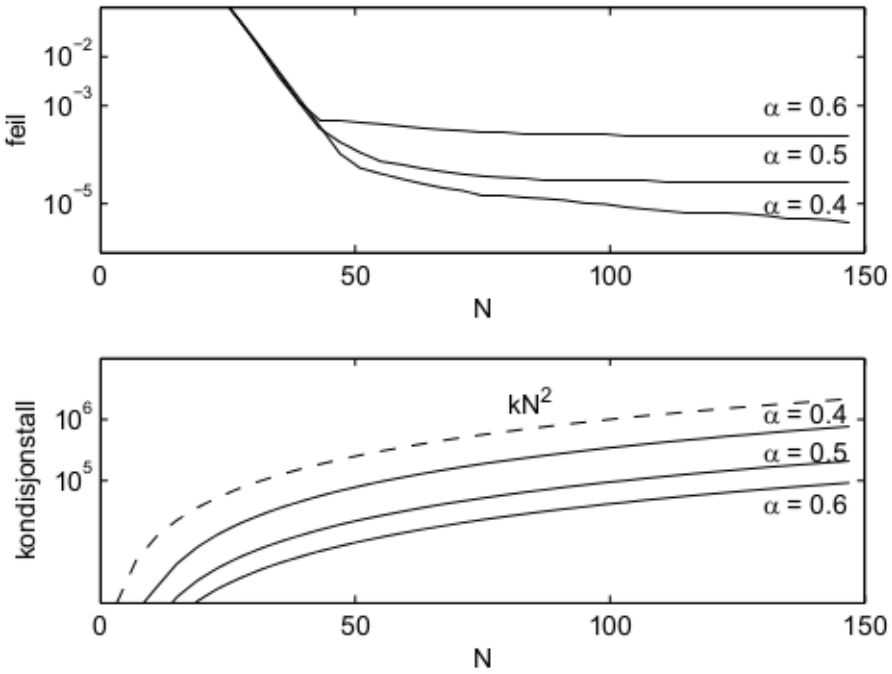
met. Formparameteren relativt til senteravstanden vil da ikke lenger være konstant. Det blir dermed naturlig å definere en individuell formparameter til hver basisfunksjon; $\epsilon_i = \alpha/h_i$. Her er h_i er avstanden mellom basisfunksjonen i sitt senter og dens nærmeste senter. Fremdeles velges en konstant α for å finne en god 'startverdi' til formparametrene.

Figur 2.8 viser approksimasjonsfeil og kondisjonstall til Poisson 1D ved å bruke chebyshevpunktene som senterfordeling. Konvergensraten er omtrent lik som for testen med uniform senterfordeling (Figur 2.6), men kondisjonstallet øker nå algebraisk.

Den adaptive metoden velger formparametrene individuelt som funksjon av senteravstaden. Dette tyder på at algebraisk økning av kondisjonstallet er hva man kan forvente for den adaptive kollokasjonsmetoden. Ideelt burde kondisjonstallet vært uavhengig av antall sentre som tilfellet er for relativ formparameter på uniform senterfordeling. Algebraisk økningsrate er likevel et bedre resultat enn eksponensiell økning som forekommer ved å bruke en konstant formparameter som ikke er definert relativt til senteravstanden (se figur 2.4).

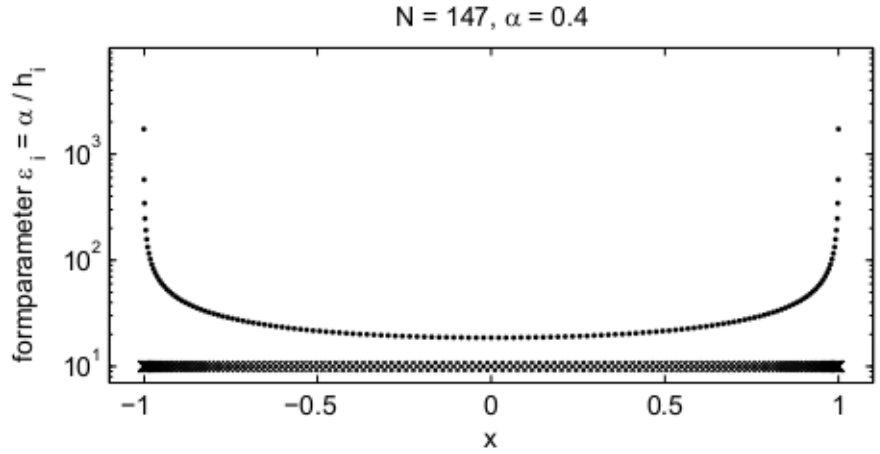
Figur 2.9 viser størrelsene på formparametrene når de defineres individuelt etter senteravstanden. Her brukes Chebyshev senterfordeling, en senterfordelingen med økende tetthet mot rendene. Dermed blir i dette tilfellet formparametrene til sentrene nær randen omtrent hundre ganger større enn formparametrene til sentrene omkring midten av domenet.

Dette er et eksempel på en senterfordeling som ikke gjenspeiler problemet som skal approksimeres. Poisson 1D har størst variasjon omkring senteret på domenet. Med en adaptiv senterfordeling vil det være naturlig med en høyere



Figur 2.8: *Approksimasjonsfeil og kondisjonstall for Chebyshev senterfordeling med formparametre relativt til senteravstanden.*

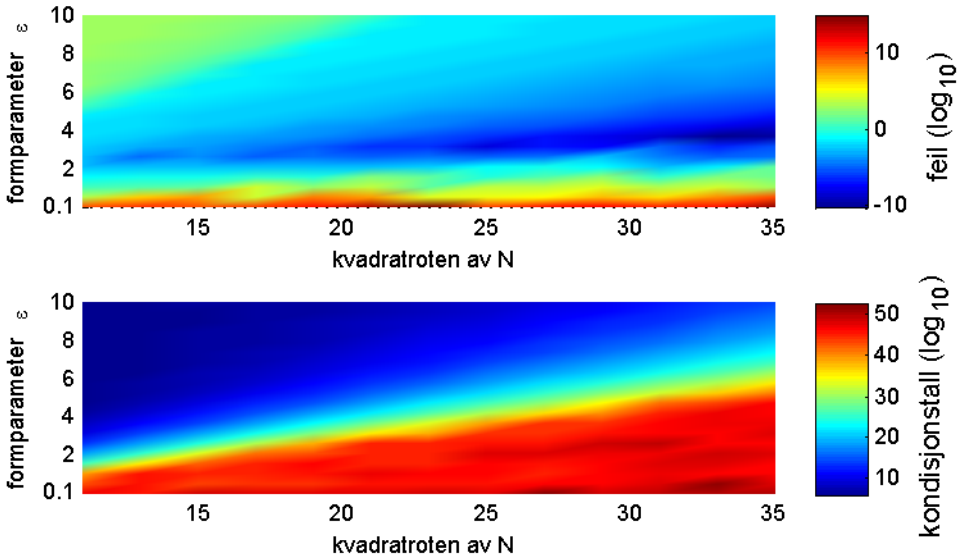
senterfordeling omkring dette området.



Figur 2.9: *Verdiene til de individuelle formparametrene. Kryssene indikerer sentre, mens prikkene er verdien på de individuelle formparametrene.*

Flere dimensjoner

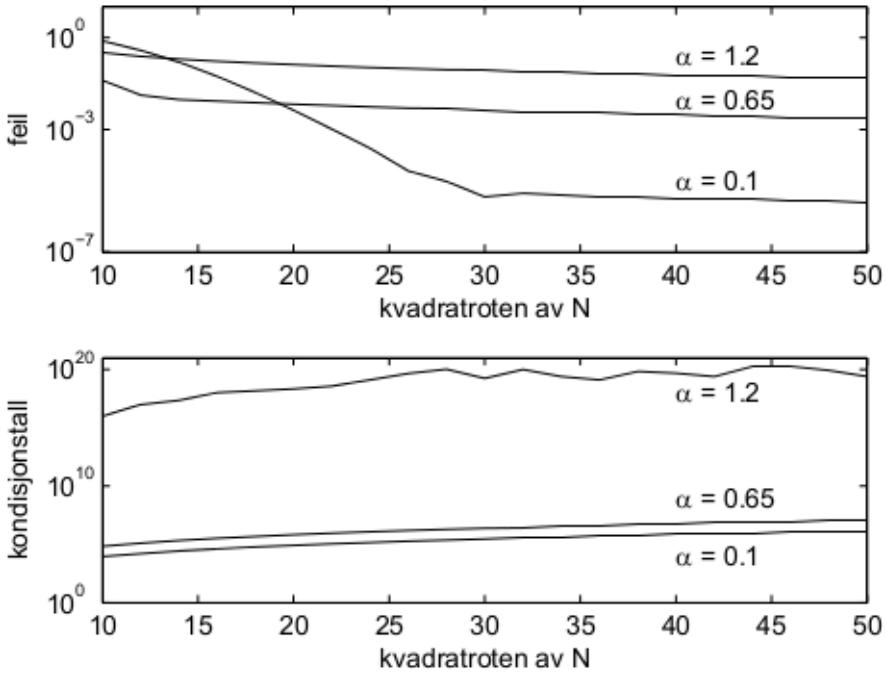
Forsøkene som er beskrevet over kan også gjøres på flerdimensjonale problemet. Figur 2.10 viser '$\epsilon - N$' landskapet til poisson 2D: $u_{xx}+u_{yy} = f$ på domenet $[-1, 1]^2$ med Dirichlet randkrav og uniformt fordelte sentre. Modellproblemet blir laget med utgangspunkt i den eksakte løsningen $u = \frac{1}{(25x^2+1)(25y^2+1)}$



Figur 2.10: *Approksimasjonsfeil med kondisjonstall til poisson 2D med varierende formparameter og økende antall sentre*

Figuren viser den samme trenden som forsøket med poisson 1D (se figur 2.5), nemlig at den største trusselen for nøyaktigheten ved økende antall sentre er det stigende kondisjonstallet.

Tilsvarende som for Poisson 1D kan formparameteren defineres relativt til senteravstanden. Figur 2.11 viser resultatet av dette når sentrene er uniformt fordelt i det todimensjonale domenet. Konvergensraten er ikke like god som for det tilsvarende forsøket med Poisson 1D (Figur 2.6), kondisjonstallet er heller ikke uavhengig av antall sentre. Dette er likevel som forventet, ettersom det er en langt mer komplisert interaksjon mellom nærliggende sentre i to dimensjoner.



Figur 2.11: *Kollokasjon av Poisson 2D med konstant formparameter definert relativt til nærmeste senter*

Kapittel 3

RBF adaptiv kollokasjonsmetode

Radielle basisfunksjoner har som kjent opphav i interpolasjon av spredte datapunkter. Plasseringen av interpolasjonspunktene er da bestemt av datasettet. For kollokasjonsproblemer er situasjonen annerledes. Det finnes ikke lenger en forhåndsgitt fordeling av kollokasjonspunktene.

Selv om radielle basisfunksjoner gir stor frihet i hvor kollokasjonspunktene skal plasseres, burde ikke denne fordelingen være tilfeldig. Hensikten med den adaptive kollokasjonsmetoden er å fordele basisfunksjonene på en måte som gjenspeiler forhold ved selve problemet, men som også tar stabilitetsmessige hensyn ved å velge formparametrene på en måte som holder kondisjonstallet lavt.

Behovet for en datastruktur

Gode og effektive algoritmer kjennetegnes ofte ved gode datastrukturer. En god datastruktur bør være effektiv, robust og fleksibel med tanke på datamengden den skal organisere. Dette blir spesielt viktig for adaptive metoder hvor sentre blir lagt til eller fjernet, noe som fører til ytterlige utfordringer med tanke på organisering av datamengden.

Det er ikke mye å oppdrive i litteraturen når det gjelder beskrivelser av datastrukturer for hvordan sentrene med tilhørende data skal fordeles for RBF-metoder. Årsaken kan være at radielle basisfunksjoner for kollokasjonsmetoder fremdeles befinner seg i utviklingsfasen. Det finnes få anvendelser

utover relativt enkle modellproblemer hvor fokus er på numeriske egenskaper fremfor 'industriell' robusthet.

I forbindelse med adaptive RBF metoder finnes det noen beskrivelser av datastrukturer. J. Behrens [2] benytter seg av en Delaunay triangulering og dens duale Voronoi diagram for å legge til nye sentre. Dette er velkjente verktøy blant annet for triangulering av domenet i endelig elementmetoder.

Trianguleringsalgoritmer er komplekse både når det gjelder implementering og når det gjelder bruk av regnekraft. Kompleksiteten med å tilpasse trianguleringen adaptivt øker med økende dimensjoner.

En annen vei å gå når det gjelder datastruktur er i retning kd-tre datastrukturer. Dette er en gruppe velprøvde og robuste datastrukturer som oppstår naturlig ved alle typer rekursive metoder hvor en bryter et problem ned i flere deler for å gjenta samme prosess på hver del. Man finner anvendelser av denne typen datastrukturer i informatikk, bildebehandlig – til og med innen mer eksotiske felt som slektsgranskning.

Årsaken til at denne typen datastruktur blir brukt i oppgaven er at kd-tre strukturer er langt enklere å implementere enn de som baseres på triangulering. Implementert på rett måte er det også en regnekraftmessig billigere datastruktur. Datastrukturen er i tillegg enkel å generalisere til et vilkårlig antall dimensjoner. Kompleksisten med å legge til eller fjerne sentre er kun avhengig av antall noder i datastrukturen – ikke dimensjonen på problemet som er tilfellet for trianguleringsstrukturer.

I tillegg finnes det mange eksempler på bruk av kd-tre på adaptive kollokasjonsmetoder, mest utprøvd er endelig differansemetoder. Relevant for denne oppgaven er at datastrukturen også er forsøkt benyttet på RBF adaptive metoder [14, 6]. Artikkelen til Driscoll [6] er utgangspunktet for den adaptive metoden i denne oppgaven.

Nøyaktigheten til RBF-metoder er vel så avhengig av hvordan man velger formparameteren ϵ som antall sentre N i approksimasjonen. Kd-tre datastrukturen brukes til å gi hver basisfunksjon en individuell formparameter som funksjon av nodens nivå i treet.

Annen viktig informasjon som kan håndteres av en kd-tre datastruktur er om senteret befinner seg på randen, en indikator som avgjør om det behøves flere sentre i området, samt områdets areal.

En svakhet som bør nevnes ved boksbaserte rekursive metoder gjelder håndteringen av domener med irregulære domener. Her viser Delaunay triangulering langt større fleksibilitet. Som vi skal illustrere senere er det mulig

å implementere boksbaserte metoder på vilkårlig geometri ved hjelp av en geometrisk forfiningsprosess i starten av den adaptive prosessen.

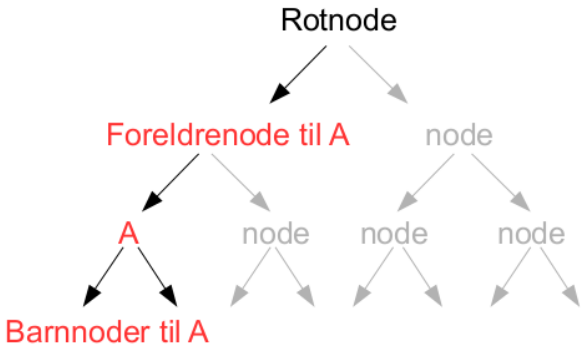
3.1 Beskrivelse av RBF adaptiv 1D

Metoden blir implementert for endimensjonale problemer for å på en enklest mulig måte kunne vise effekten av adaptivitet. Denne metoden blir senere brukt ved parametriseringen av irregulære render for todimensjonale problemer.

Om datastrukturen

Oppgaven bruker en direkte oversettelse av de engelske termene som brukes i beskrivelsen av hierarkiske datastrukturer. Generelt beskrives datastrukturen som en samling noder, hvor hver enkelt node inneholder data som er spesifikk for datastrukturen. Figur (3.1) viser et binærtre som i denne oppgaven oppstår ved partisjonering av et endimensjonelt domene.

Hver node (med unntak av rotnoden) har alltid én foreldernode og opptil to barnnoder. Dybden på treet gjenspeiler antall rekursjoner. På figur 3.1 oppstår node A etter to rekursjoner med utgangspunkt i rotnoden.



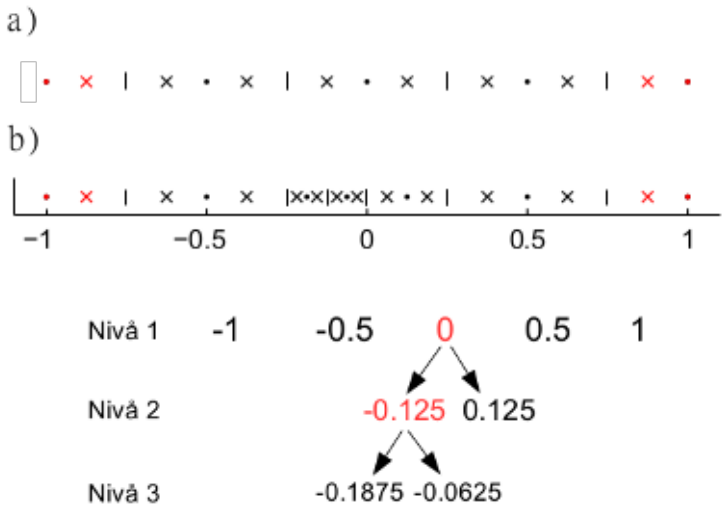
Figur 3.1: *Eksempel på et binærtre hvor hver node har opptil to barn.*

Metoden er en endimensjonal analogi til den todimensjonale varianten som beskrives i 'Residual subsampling methods for radial basis function interpolation and collocation problems' [6].

Metoden initieres ved å partisjonere domenet i et gitt antall subintervaller som til sammen dekker domenet. Midten av hvert intervall tildeles et senter

samt to sjekkpunkt, se figur 3.2 a). Et unntak gjelder de to intervallene som grenser til randpunktene. De skiller seg fra de indre intervallene ved å være av halv lengde, og kun inneha ett sjekkpunkt.

Figur 3.2, b) viser en hypotetisk senterfordeling etter to iterasjoner i den adaptive prosessen. Det respektive binærtreet vises også. Merk at termen *node* ikke brukes som kollokasjonspunkt, men som datareferanse i kd-treet.



Figur 3.2:
a) Initiell partisjonering av domenet i tre indre intervall, og to randintervall (i rødt). Prikk (·) markerer senter. Kryss (x) er sjekkpunkt.
b) Tenkt situasjon etter to rekursjoner med tilhørende trestruktur. Her er sjekkpunktene i det opprinnelige senterintervallet omgjort til sentre i nye intervaller. Deretter er sjekkpunktene i ett av de nye intervallet er omgjort til nye sentre. Merk at sentrene $x = 0$ *og* $x = -0.125$ *ikke brukes som kollokasjonspunkt.*

Sjekkpunktene spiller en sentral rolle i den adaptive prosessen. Hvert sjekkpunkt blir tilegnet en tallverdi kalt *indikator*. Disse verdiene avgjør om intervallet skal forfines til to intervall med halv lengde, eller om intervallet skal fjernes til fordel for et større intervall av dobbel lengde.

For poissonproblemer settes indikatorverdien i sjekkpunktet i lik residualen

$$R^i = Lu_N^i - f^i \tag{3.1}$$

hvor f^i er en funksjonsverdi fra høyresiden til poissonproblem. Substitueres $Lu^i = f^i$ i (3.1) har vi

$$\boldsymbol{r} = \boldsymbol{D}\boldsymbol{u}_N - L\boldsymbol{u}$$

hvor $\boldsymbol{r}$ og $\boldsymbol{u}_N$ er vektorer, $\boldsymbol{D}$ er derivasjonsmatrisen og $L\boldsymbol{u}$ en vektor bestående av eksakte verdier i sjekkpunktene. Ved å diskretisere L, $L\boldsymbol{u} \approx \boldsymbol{D}\boldsymbol{u}$, har vi

$$\begin{aligned} \boldsymbol{r} &\approx \boldsymbol{D}\boldsymbol{u}_N - \boldsymbol{D}\boldsymbol{u} = \boldsymbol{D}(\boldsymbol{u}_N - \boldsymbol{u}) \\ \boldsymbol{D}^{-1}\boldsymbol{r} &\approx \boldsymbol{u}_N - \boldsymbol{u} \end{aligned}$$

som ved å ta normen og bruke Cauchy-Schwarz ulikheten gir følgende uttrykk for approksimasjonsfeilen

$$\|\boldsymbol{u_N} - \boldsymbol{u}\| \approx \|\boldsymbol{D}^{-1}\boldsymbol{r}\| \leq \|\boldsymbol{D}^{-1}\|\|\boldsymbol{r}\|$$

Residualen brukes til å lokalisere områder i domenet hvor det trengs flere eller færre kollokasjonspunkt. Områder med høy residualverdi tolkes som områder med høy approksimasjonsfeil, og dette bøtes på ved å sette inn flere kollokasjonspunkt i området. I områder med lav residualverdi kan kollokasjonspunkt fjernes.

Hvordan residualen fordeler seg er spesifikt for hvert enkelt problem, og indikatoren fungerer dermed som et forsøk på å lage en senterfordeling som er problemavhengig.

I tillegg tilpasses formparameteren individuelt til hver basisfunksjon. På bakgrunn av observasjonene gjort i forrige kapittel må formparameterne justeres i takt med at nye sentre fjernes eller legges til. I områder med høy sentertetthet økes basisfunksjonenes formparameter, mens i områder med lav sentertetthet kan formparametrene holdes lavere. Tabell 3.1 oppsummerer disse grunnleggende prinsippene.

Tabell 3.1: ***Prinsipper for adaptiv senterplassering og formparametre***

Høy residualverdi	$\Rightarrow$	sette inn nytt senter
Lav residualverdi	$\Rightarrow$	fjerne eksisterende senter
Høy sentertetthet	$\Rightarrow$	øke formparameter ('bratte' RBF)
Lav sentertetthet	$\Rightarrow$	minke formparameter ('flate' RBF)

Strategi for senter- og epsilonfordeling

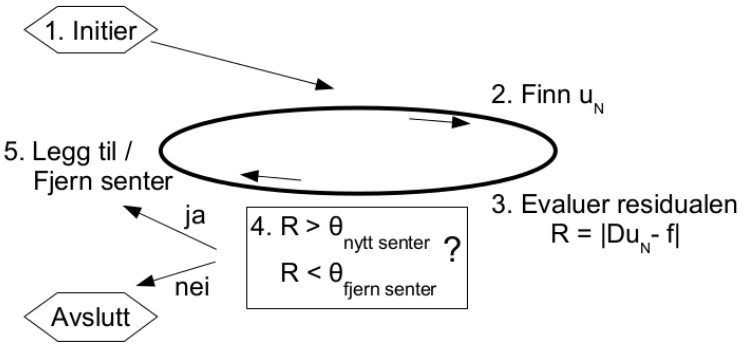
Indikatorverdiene til sjekkpunktene er utslagsgivende på om sjekkpunkt blir omgjort til nye sentre, eller om eksisterende sentre blir fjernet og omgjort til sjekkpunkt.

Det settes en øvre terskel (kalt *ovre* i påfølgende kapittel). Hvis *begge* sjekkpunktenes indikatorverdier til et intervall overstiger denne, blir sjekkpunktene omgjort til nye sentre i intervall med halvparten av den opprinnelige lengden. Formparametrene til de nye sentrene blir satt til det dobbelte av formparameteren til det opprinnelige senteret. Det opprinnelige senteret blir fjernet. Dermed har man en netto økning på ett senter for hvert intervall som blir forfinet.

I tillegg settes det en nedre terskel (kalt *nedre* i påfølgende kapittel). Hvis indikatorverdiene til *begge* sjekkpunktene i intervallet er mindre enn denne verdien, fjernes intervallet. Senteret som befinner seg på foreldernoden i binærtreet blir reaktivert. Formparameteren tilhørende dette senteret blir satt til halvparten av verdien til senteret i intervallet som fjernes.

Metodens syklus

Metodens sykliske prosess oppsummeres i figur 3.3.



Figur 3.3: *Diagram som illustrerer stegene i den adaptive syklusen.*

1. Metoden initieres ved å dekke domenet med N antall intervall (dermed også N sentre). Intervallene utgjør et minste antall sentre som ikke kan fjernes senere i den adaptive prosessen. I tillegg spesifiseres det en felles formparameter for sentrene. Øvre og nedre terskel for å henholdsvis forfine og fjerne intervall spesifiseres også.

2. Nødvendige data for å sette sammen kollokasjonsmatrisen hentes ut av binærtreet. Kollokasjonsmatrisen settes sammen på formen $\boldsymbol{K\lambda} = \boldsymbol{b}$ etter Kansas metode som beskrevet i forrige kapittel.

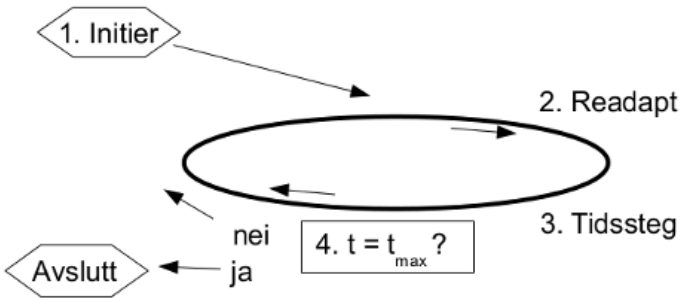
3. Residualen evalueres i sjekkpunktene $y_1, ..., y_m$ med formelen $\boldsymbol{R} = |\boldsymbol{E\lambda} - \boldsymbol{f}|$. Her er $\boldsymbol{E}$ evalueringsmatrisen (2.5), $\boldsymbol{\lambda}$ koeffisientvektoren til basisfunksjonene og $\boldsymbol{f}$ høyresiden i poissonligningen.

4. Dette er det adaptive steget i syklusen. Intervall legges til eller fjernes som beskrevet i strategien for senter- og epsilonfordeling. I tillegg vurderes et vilkår for å avslutte syklusen. Syklusen avsluttes når alle sjekkpunktene er under den øvre grensen satt for residualen, det vil si når det ikke legges til nye sentre i prosessen.

5. Nye intervall som bestemt i forrige steg legges til eller eksisterende intervall fjernes. Prosessen går så inn i en ny syklus.

Adaptiv RBF 1D med linjemetoden

Med relativt små modifikasjoner kan den samme adaptive metoden implementeres for differensialproblemer på formen $u_t = Lu$ ved hjelp av linjemetoden.



Figur 3.4: *Diagram over adaptiv RBF med linjemetoden*

1. I tillegg til parametrene fra adaptiv RBF for poissonproblemer spesifiseres det en steglengde Δt. Det angis også en sluttid t_{max} for å avslutte prosessen. I tilfeller hvor det velges eksplisitt tidslinjemetode, angis i tillegg parameteren Δt_{adapt} for tidsintervallet hvor readapteringen skjer.

2. Denne prosessen er identisk med syklusen beskrevet over, men med ett viktig unntak. Mens indikatorfunksjonen for poissonproblemer er basert på residualen, brukes det i dette tilfellet en indikatorfunksjon basert på interpolasjon.

   Ved første readaptering $t = 0$ brukes indikatorfunksjonen $P = |u_N - u^0|$. Her er u_N den approksimerte løsningen ved tiden $t = 0$ og u^0 er startverdiene som spesifiseres av problemet.

Ved tidssteget t interpoleres den approksimerte løsningen fra det forrige tidssteget u_N^{t-1} til sjekkpunktene for det aktuelle tidssteget, så sammenlignes den med verdiene for det aktuelle tidssteget u_M^t.

Indikatorfunksjonen, P, beregnes slik:

$$\boldsymbol{P} = |\boldsymbol{E}_N \boldsymbol{\lambda}_N^{t-1} - \boldsymbol{E}_M \boldsymbol{\lambda}_M^t| \tag{3.2}$$

Her er $\boldsymbol{E}_N$ (dimensjon MxN) og $\boldsymbol{E}_M$ (dimensjon MxM) evalueringsmatrisene på samme form som uttrykket (2.5) men med N sentre fra forrige tidssteg $t-1$ og M sentre fra det aktuelle tidssteget t. Koeffisientvektorene $\boldsymbol{\lambda}_N^{t-1}$ og $\boldsymbol{\lambda}_M^t$ er fra tidsstegene $t-1$ og t.

Leddet $\boldsymbol{E}_M \boldsymbol{\lambda}_M^t$ i indikatorfunksjonen blir oppdatert etter hver syklus i readapteringen, og M vil derfor øke etter hvert som nye sentre blir lagt til.

Differansen mellom to interpolanter vil som regel gi større utslag i områder med høy variasjon og mindre i områder med lav variasjon. Etter hvert som nye sentre legges til eller fjernes vil M nærme seg N, og utslagene til P blir mindre. Når indikatorverdiene til sjekkpunktene befinner seg i intervallet definert av øvre og nedre terskel, avsluttes readapteringen.

Merk at indikatorfunksjonen på ingen måte gjenspeiler den virkelige approksimasjonsfeilen. Derimot viser den seg å gjøre jobben med tanke på de enkle prinsippene om senterfordeling som den adaptive metoden bygger på.

3. Med utgangspunkt i koeffisientvektoren $\boldsymbol{\lambda}_M$ settes derivasjonsmatrisen $\boldsymbol{D} = \boldsymbol{A_L} \boldsymbol{A^{-1}}$ sammen. Den approksimerte løsningen flyttes så et steg Δt i tid. Metoden er forsøkt implementert med både Runge Kutta fjerde ordens eksplisitt metode og MATLABs implisitte *ode15s* løser.

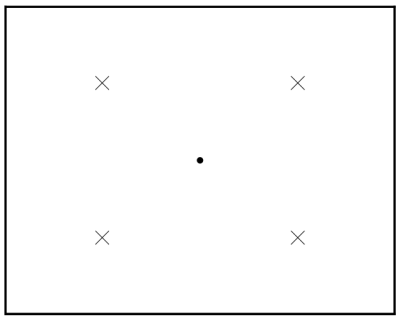
   I tilfellet med *ode15s* integreres løsningen et 'steg' Δt_{adapt} mellom hver readaptering. Det vil si for hver syklus finnes u^{t+1} med *ode15s* ved å velge u^t som initialverdier.

4. Syklusen gjentas og avsluttes når den oppgitte sluttiden t_{max} er nådd.

3.2 Beskrivelse av RBF adaptiv 2D

Mekanismene for den todimensjonale adaptive metoden er tilsvarende som for i én dimensjon med unntak av forhold som har med geometrien å gjøre.

Mens det endimensjonale domenet blir delt i intervaller, blir det todimensjonale delt i todimensjonale intervaller kalt bokser. Hver boks inneholder ett senter og fire sjekkpunkt som vist på figur 3.5 (et unntak er tilfellet med irregulære render).



Figur 3.5: *Boks med senter (·) og fire sjekkpunkt (x)*

Det rektangulære domenet dekkes med et lappeteppe av slike bokser. Den hierarkiske datastrukturen som oppstår ved å rekursivt omgjøre sjekkpunkt til sentre i nye bokser kalles kvadtre. I motsetning til binærtreet hvor hver node kan ha opptil to barn, kan hver node i kvadtreet ha opptil fire barn.

Strategi for senter- og epsilonfordeling

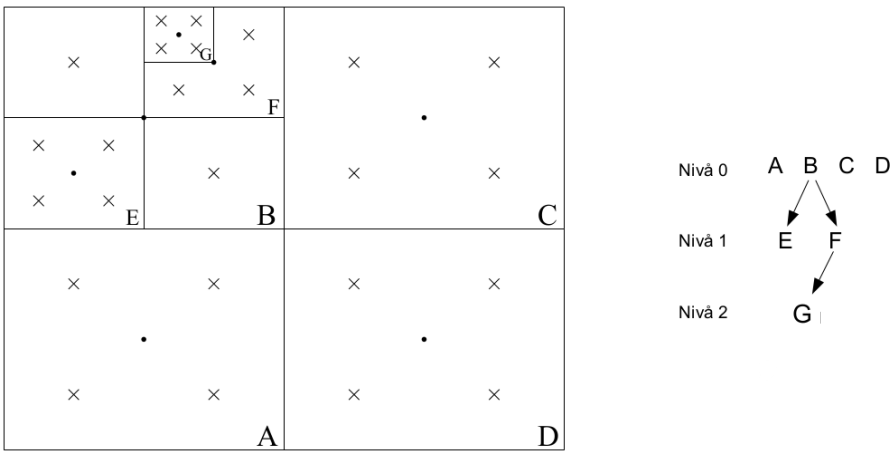
I motsetning til adaptiv RBF 1D blir nå et sjekkpunkt med indikatorverdi over forfiningsterskelen omgjort til et nytt senter som er uavhengig av de resterende sjekkpunktene i boksen. Dette gjør at senterfordelingen raskt kan økes i et lokalisert område med høy indikatorverdi, selv med en grov initiell boksinndeling.

Et senter fjernes hvis indikatorverdiene til alle fire sjekkpunktene i boksen er under forgrovingsterskelen. Dette valget er primært av implementeringsmessige hensyn, for å opprettholde konsistensen på det tilhørende kvadtreet.

Formparametrene tilpasses også etter regelen om at den velges til det dobbelte av forelderens epsilonverdi. I motsetning til for adaptiv RBF 1D velges det å beholde sentrene i boksene selv hvor alle sjekkpunktene er omgjort til nye sentre.

Figur 3.6 viser en mulig bokskonfigurasjon etter to iterasjoner. Domenet bestod opprinnelig av fire bokser A,B,C,D. I første iterasjon ble det nordøstlige og sørvestlige sjekkpunktet til boks B omgjort til sentre i nye bokser E og

F. I andre iterasjon ble det nordvestlige sjekkpunktet i boks F omgjort til senter i en ny boks G.



Figur 3.6: *Dekomponert område med tilhørende trestruktur*

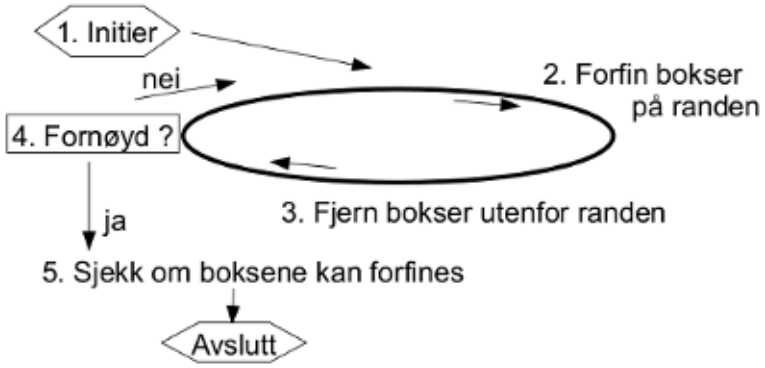
Geometrisk områdeforfining

Et av argumentene til fordel for en gitterfri metode er at den håndterer irregulær geometri. Dette bør også gjelde for den adaptive metoden.

Driscoll og Heryudono [6] foreslår en måte å håndtere vilkårlig geometri for et todimensjonalt domene. Vi følger deres ideer, men anvender en litt forenklet strategi for forfining nær randen.

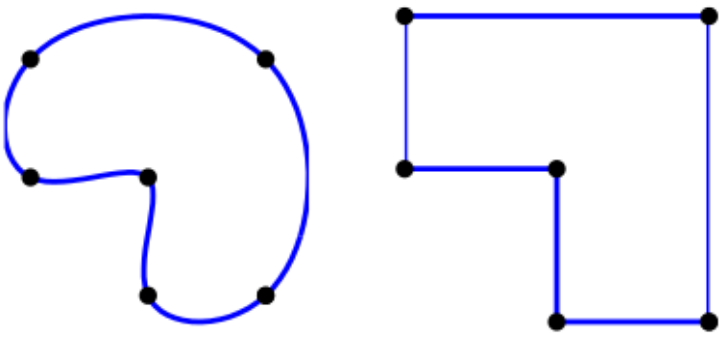
Geometrisk områdeforfining kjøres i forkant av den adaptive prosessen. Hensikter er, med utgangspunkt i et domene bestående av bokser, å avgjøre hvilke som befinner seg innenfor domenet og som dermed kan brukes i den adaptive prosessen. Bokser som befinner seg utenfor domenet er ikke egnet i den adaptive prosessen og blir derfor forkastet.

Beskrivelse av prosessen



Figur 3.7: *Geometrisk forfiningssyklus*

1. Randen er en parametrisert utgave av RBF adaptiv 1D. Randen defineres av en punktmengde som interpoleres enten med delvis lineære polynomer eller med polynomer hvor i tillegg opp til polynomets andrederiverte er kontinuerlig (kubisk spline). Figur 3.8 viser forskjellene på domenene som defineres på denne måten.

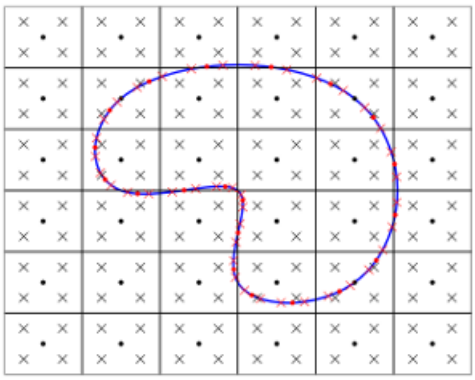


Figur 3.8: *Spline interpolasjon av seks noder (venstre). Delvis lineær interpolasjon av samme noder (høyre)*

   Randen plasseres så på et 'lappeteppe av bokser' som vises på figur 3.9.

2. Bokser som skjærer randen blir forfinet, det vil si at samtlige av boksens sjekkpunkt blir omgjort til sentre i nye bokser.

   Nye bokser som i sin helhet befinner seg innenfor randen blir godtatt. De blir så slettet fra datastrukturen og sjekkpunktene som gir opphav



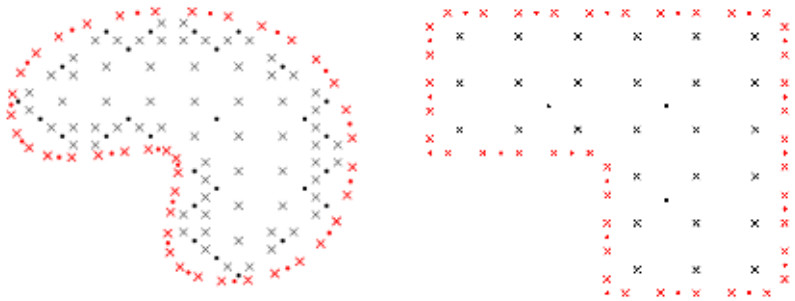
Figur 3.9: *Parametrisert rand på lappeteppe av bokser.*

til boksene blir godtatt.

Nye bokser som skjærer randen blir merket til neste runde av prosessen.

3. Nye bokser som i sin helhet befinner seg utenfor randen blir forkastet.
4. Ut i fra hvilken 'oppløsning' som ønskes kan steg 2 og 3 gjentas.
5. Før syklusen avsluttes må boksene som skjærer randen kontrolleres. Sjekkpunkt og sentre som befinner seg utenfor randen merkes som ubrukelige i datastrukturen. Av gjenværende sjekkpunkt og sentre kontrolleres det for om de gir opphav til nye bokser som skjærer randen. I så fall forkastes de også.

Figur 3.10 viser resultatet av forfiningsprosessen.



Figur 3.10: *Senter (·) og sjekkpunkt (x) etter den geometriske forfiningsprosessen for domener definert av spline interpolasjon (venstre) og lineær interpolasjon (høyre). Randsentre og randsjekkpunkt i rødt.*

Kapittel 4

Resultater fra RBF adaptiv

I denne oppgaven fokuseres det på å undersøke effekten av adaptiv plassering av sentrene med RBF kollokasjon. Vi har beskrevet tre ulike situasjoner hvor adaptivitet potensielt kan gi en bedre approksimasjon:

1. Problemer hvor løsningen varierer mye i et lokalisert område av domenet. I slike områder tilsier intuisjonen det er fordelaktig med en høyere sentertetthet, mens vi kan ha færre senterpunkt i periferien av området. Dermed oppnås en mer effektiv senterfordeling.

2. Problemer hvor den lokaliserte løsningen forflyttes i tid. For denne typen problemer er fokus på å teste om metoden klarer å fange opp slike forflytninger og om griddet blir tilpasset for hvert tidssteg.

3. I realistiske simuleringer benyttes ofte irregulær geometri. Radielle basisfunksjoner klarer dette uproblematisk. Her er derimot fokus på å teste om den adaptive metoden og dens datastruktur har den nødvendige fleksibiliteten.

I tråd med eksperimentell praksis har vi fokusert på enkle modellproblemer som kan brukes til å si noe metodens numeriske egenskaper. Kun mindre modifikasjoner skal til for å håndtere mer realistiske problemer med variable koeffisienter og korrekt skalerte enheter.

For punkt 1 og 2 er modellproblemer i én romlig dimensjon tilstrekkelig for å kunne si noe den adaptive metodens effektivitet. For punkt 3 er det nødvendig å øke til to dimensjoner.

Merk at den adaptive metoden er generaliserbar til flere dimensjoner, men slike undersøkeler er utenfor rammen til denne oppgaven.

Implementeringsmessig har det vært fokus på å gjøre koden enkel og oversiktelig fremfor smart og rask.

Kd-tre datastrukturene har blitt implementert på en 'naiv' måte. Kompleksiteten på å hente ut data, sette inn nye noder samt slette noder er alle av størrelsesorden $\mathcal{O}(N)$. En mer effektiv implementering kan redusere kompleksiteten for å sette inn og slette noder til størrelsesorden $\mathcal{O}(\log N)$.

Det viste seg at den største andelen av kjøretiden gikk med til operasjoner på datastrukturen. Teoretisk sett er matriseinverteringen den desidert største utgiftsposten med en kompleksitet på $\mathcal{O}(N^3)$. Det er ingen hemmelighet at prekompilerte funksjoner i MATLAB utnytter prosessorkraften langt mer effektivt. Det finnes programpakker for å kompilere MATLAB skript til C/C++ kildekode, noe som trolig ville minket kjøretiden til datastrukturen.

For å initiere RBF adaptiv må en rekke parametre spesifiseres. Parameteren N angir antall initielle sentre (i 1D er N antall intervall, i 2D antall bokser). Parameteren ϵ angir formparameteren til de initielle sentrene. Parametrene *ovre* og *nedre* angir terskel for å henholdsvis legge til eller fjerne sentre. For RBF adaptiv med linjemetoden angis Δt som er tidsintervallet mellom hver readaptering av sentrene.

Parameterne velges ut fra strategien om å holde antall sentre på et lavt nivå. Erfaringene fra kapittel 2 tilsier at konvergensraten er høyest når antall sentre er lavt. I tillegg er forsøkene fra litteraturen som RBF adaptiv sammenlignes med, utført med et lavt antall sentre.

Altså velges antall initielle sentre N til et lavt tall. Det gjør også formparameteren ϵ, men uten at det medfører at kondisjonstallet blir for høyt. Terskelen *ovre* settes høyt for at metoden skal konvergere på et lavt antall iterasjoner. Terskelen *nedre* settes typisk til $ovre^2$. Om *nedre* velges for høyt vil det føre til 'oskillasjoner' hvor mange sentre blir fjernet for deretter bli lagt til i neste iterasjon.

4.1 Resultater fra RBF adaptiv 1D

Poisson modellproblemer

Dette er en elliptisk partiell differensialligning på formen

$$\Delta u = f \tag{4.1}$$

Poissonligningen er et viktig modellproblem i anvendt matematikk. Den brukes blant annet til å modellere flere fenomener innen fysikken. Funksjonen u kan innen elektrostatikk være potensialet i elektrisk felt, eller u kan være profilen til en membran som er festet i rendene mens det blir utsatt for et kraftfelt f.

I én romlig dimensjon uttrykkes poissonligningen som

$$u_{xx}(x) = f(x), \quad x \in (-1, 1) \tag{4.2}$$

her er $f(x)$ en kjent funksjon mens $u(x)$ er funksjonen som skal approksimeres. I tillegg spesifiseres Dirichlet randkrav

$$u(-1) = g_{-1}$$
$$u(1) = g_1$$

Det første modellproblemet som forsøkes er med utgangspunkt i Runges funksjon $u(x) = \frac{1}{25\,x^2+1}$ som løser poissonproblemet (4.2):

$$f(x) = \frac{50\ (75\,x^2 - 1)}{(25\,x^2 + 1)^3}, \quad g_{-1} = g_1 = 1/26 \tag{4.3}$$

RBF adaptiv initieres med $N = 3$ og $epsilon = 0,1$. Øvre og nedre terskel for residualen settes til $ovre$=1e-07 og $nedre$=1e-16.

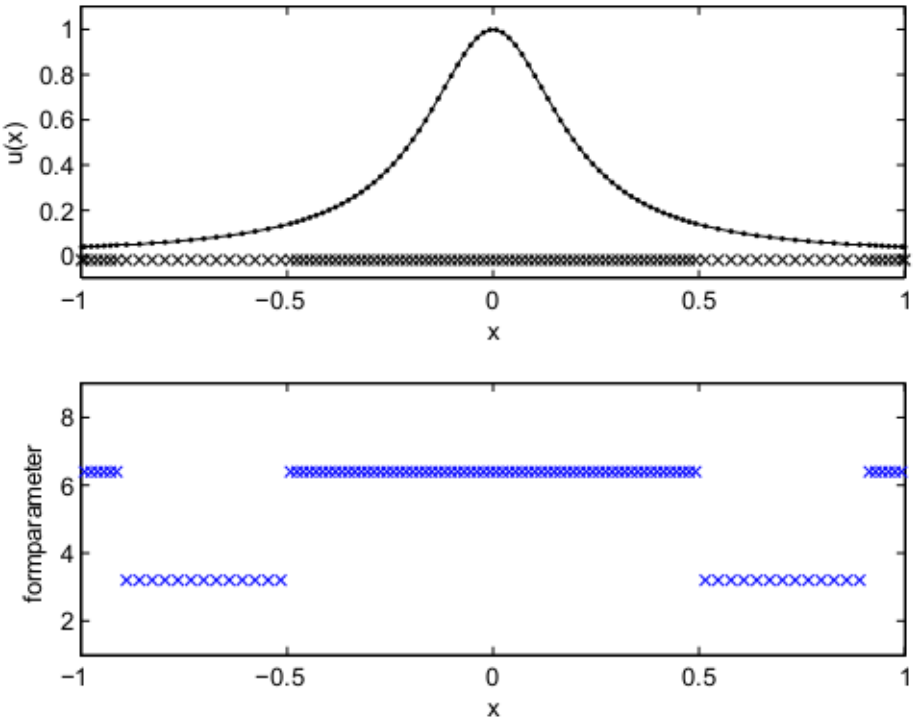
Tabell 4.1 viser at metoden konvergerer etter 6 iterasjoner. Startparametrene medfører en prosess hvor samtlige sentre i de første iterasjonene blir byttet ut mot en finere senterfordeling. Når iterasjonen avsluttes benyttes det kun sentre fra de de to siste iterasjonene. Iterasjonen kan også sees på som en form for optimeringsprosses hvor få, men flate basisfunksjoner stadig byttes ut til fordel for flere og brattere funksjoner.

Mellom iterasjon 4 og 5 skjer det noe interessant. Nøyaktigheten øker med en faktor 10^{-5} uten at kondisjonstallet øker nevneverdig. At nøyaktigheten øker uten tilsvarende økning i kondisjonstall har ikke blitt observert med konstant formparameter. Dette er samsvarende med [11] som beskriver en metode med varierende formparametre, og viser at dette fører til høyere nøyaktighet og stabilitet enn med konstant formparameter.

Figur 4.1 viser den approksimerte løsningen samt senter- og formparameterfordelingen når iterasjonen av avsluttet. Det er en høyere sentertetthet nær endepunktene samt i området hvor funksjonen varierer mest. Her er også formparameterne større, noe som vil si 'brattere' basisfunksjoner.

Iterasjon	N	N_f	N_l	$\|\cdot\|_\infty$	kondisjonstall
0	3	1	4	2,38e+01	1,54e+04
1	6	4	8	4,37e+00	3,43e+07
2	10	8	16	2,28e+00	7,54e+08
3	18	16	32	4,31e-02	9,57e+10
4	34	32	64	5,65e-04	1,16e+12
5	66	38	76	7,57e-09	2,40e+12
6	104	0	0	1,65e-12	3,97e+14

Tabell 4.1: *Resultater fra poissonproblemet (4.2) med utgangspunkt i den kjente løsningen* $u(x) = \frac{1}{25\,x^2+1}$. N_f *og* N_l *er antall sentre som henholdsvis fjernes og legges til før neste iterasjon.*

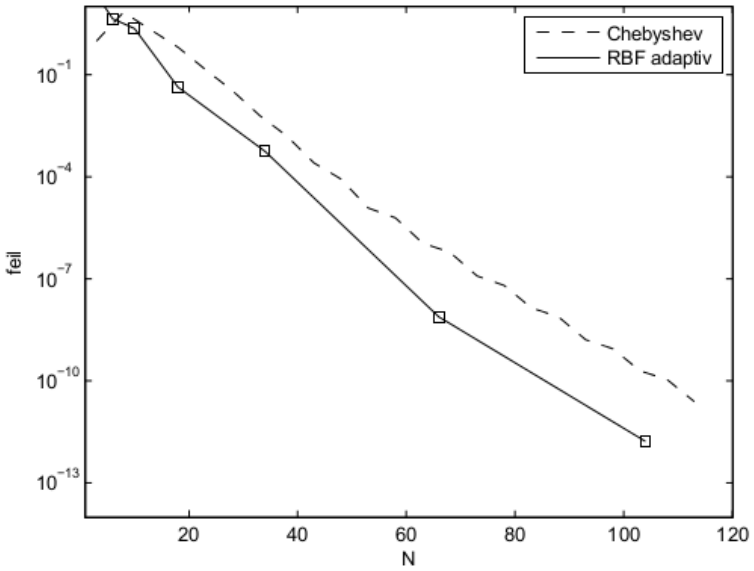


Figur 4.1: *Plot av approksimert løsning med sentre (øverst). Plot av formparametre (nederst)*

Senter- og formparameterne er fordelt i tråd med det som var hensikten, nemlig det at høyere sentertetthet i området med stor variasjon vil bidra til å fange opp mer informasjon for en mer nøyaktig approksimasjon. I tillegg kompenseres det for avveiningsprinsippet ved å øke basisfunksjonenes formparametre i områder hvor sentrene står tettere.

Det observeres høyere sentertetthet nær endepunktene i domenet. Dette gir assosiasjoner til Chebyshev punktfordelingen som et grep mot oscillasjoner nær rendene ved polynominterpolasjon. Det er kjent at de største approksimasjonsfeilene med radielle basisfunksjoner også er lokalisert nær rendene [8], og et tiltak mot dette er å øke sentertettheten nær rendene.

Figur 4.2 viser at metoden oppnår en omtrentlig eksponensiell konvergensrate. Vi ser at RBF adaptiv gjør det litt bedre enn Chebyshev.



Figur 4.2: *Figuren viser eksponensiell konvergensrate tilsvarende som for Chebyshev pseudospektralmetoden.*

Det andre modellproblemet som testes på poissonligningen lages med utgangspunkt i den eksakte løsningen

$$u(x) = \tanh(\eta x) \tag{4.4}$$

Denne funksjonen skiller seg fra Runges funksjon ved å nærme seg en diskontinuitet i $x = 0$ når parameterverdien $\eta \to \infty$. Ved å øke η økes gradienten omkring $x = 0$ til den analytiske løsningen.

Funksjonen (4.4) løser poissonligningen (4.2) for $f(x)$:

$$f(x) = 2\eta^2 \tanh(\eta x)(\tanh(\eta x)^2 - 1) \tag{4.5}$$

Som randkrav benyttes funksjonsverdiene spesifisert av den eksakte løsningen.

RBF adaptiv initieres med $N = 33$ og $\epsilon = 5$. Øvre og nedre terskel velges til henholdsvis *ovre* =1e-03 og *nedre*=1e-07. Parameteren η i (4.4) settes til $\eta = 500$. Det gir en funksjon som kan beskrives som en analytisk skrittfunksjon med verdiene -1 når $x < 0$ og 1 når $x > 0$.

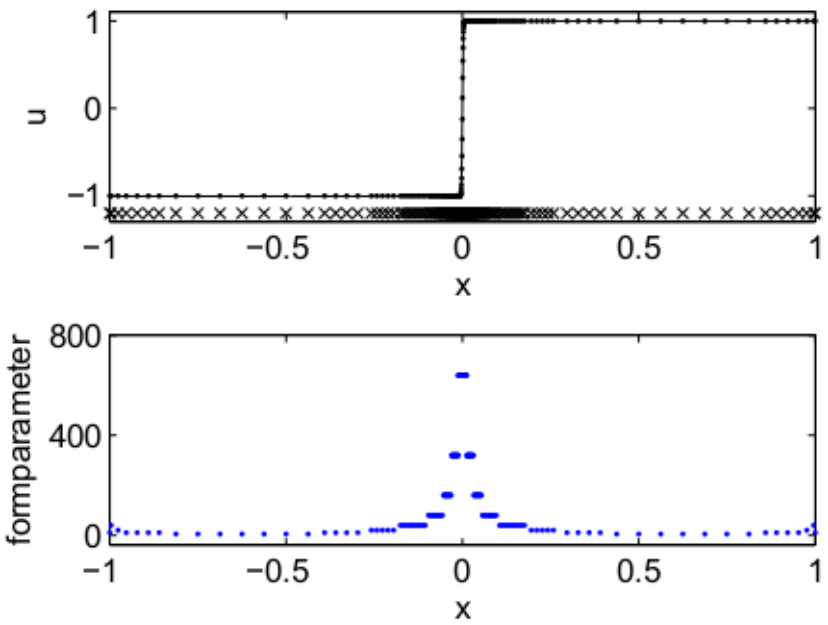
I tabell 4.2 vises resultater fra RBF adaptiv. På tilsvarende måte som for Runges testproblem trengs det noen iterasjoner før en oppnår nevneverdig nøyaktighet. Årsaken ligger i at det velges 'pessimistiske' startparametre, altså N og ϵ verdier som forventes gi lav nøyaktighet, men som har potensiale til å økes senere i iterasjonen. Merk at ved siste iterasjon blir det fjernet sentre uten at det blir lagt til nye. Likevel øker nøyaktigheten. Dette viser hvor mye formparameteren har å si for nøyaktigheten. Formparametrene til sentrene i foreldrenodene blir satt til halvparten av formparametrene til sentrene som blir fjernet. Lavere formparameter gir høyere nøyaktighet (ved lavt kondisjonstall), og dette er trolig årsaken til den uintuitive konvergensraten.

Iterasjon	N	N_f	N_l	$\|\cdot\|_\infty$	kondisjonstall
0	33	9	20	9,37e-01	2,61e+03
1	44	24	48	9,84e-01	8,28e+03
2	68	42	84	9,16e-01	2,99e+04
3	110	64	128	7,95e-01	3,14e+04
4	174	58	116	4,84e-01	1,31e+05
5	232	140	144	1,63e-02	5,73e+05
6	236	146	108	1,52e-05	2,06e+06
7	198	50	0	1,52e-07	4,08e+07

Tabell 4.2: *Resultater fra poissonproblemet med $u(x) = \tanh(\eta x)$ som analytisk løsning. N_f og N_l er antall sentre som henholdsvis fjernes og legges til før neste iterasjon.*

Figur 4.3 viser senter- og formparameterfordelingen etter den siste iterasjonen. Denne fordelingen samsvarer med forsøket med Runges poissonproblem ved at sentertettheten øker i området med høyest variasjon. Ser man på formparameterne, derimot, kommer det frem at her benyttes det sentre fra langt flere nivå fra det korresponderende binærtreet. I motsetning til for Runges funksjon hvor det kun benyttes sentre fra tilstøtende nivå, benyttes her sentre fra alle nivå i binærtreet. Årsaken er at $u(x) = \tanh(500x)$ har langt større variasjon i et mindre område enn Runges funksjon.

Resultatene fra RBF adaptiv sammenlignes med en adaptiv metode beskrevet i [14]. Begge metodene har til felles at det benyttes en kd-tre datastruktur for senterfordelingen. Den største forskjellen er at i motsetning til å bruke



Figur 4.3: *Plot av den approksimerte funksjonen og sentrene (øvre figur) til poissonproblemet med løsningen* $u(x) = \tanh(\eta x)$. *Formparametrene vises på nedre figur.*

residualen som indikator, defineres det en feilindikator

$$\eta(x) = |\tilde{u}(x) - I\tilde{u}(x)| \tag{4.6}$$

hvor $\tilde{u}(x)$ er den numeriske approksimasjonen og $I\tilde{u}(x)$ er en lokal interpolant ut i fra nærmeste nabosentre til x bortsett fra x selv.

Tabell 4.3 sammenligner nøyaktigheten med tre andre kollokasjonsmetoder. At RBF adaptiv er mer nøyaktig enn den adaptive RBF-metoden i [14] skyldes trolig at sistnevnte ikke har en innlagt tilpasning av formparameteren i områder med høy sentertetthet. Resultatene viser også fleksibiliten til adaptive metoder sammenlignet med standard RBF og Chebyshev kollokasjon. Ved å kunne tilpasse basisfunksjonene til å fange opp skarpe variasjoner i et lokalisert område oppnås høy nøyaktighet med et lavt antall sentre sammenlignet med Chebyshev.

Merk at sammenlignet med Chebyshev basisfunksjoner er i dette tilfellet situasjonen snudd på hodet stabilitetsmessig. Dersom vi bruker Chebyshev med 4000 kollokasjonspunkt blir nøyaktigheten fortsatt ikke bedre enn 1,32e-06. Kondisjonstallet til Chebyshev er da 6,64e+14. Kondisjonstallet til RBF adaptiv er 4,08+07 når approksimasjonsfeilen er 1,52e-07.

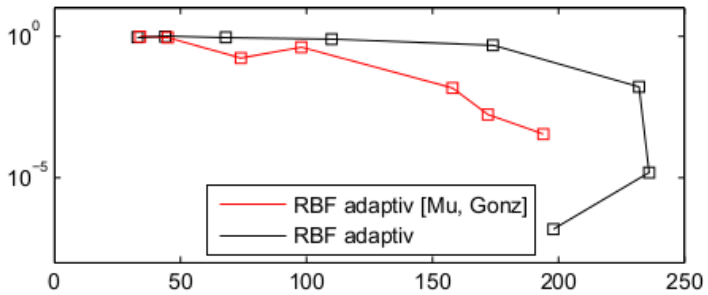
	RBF adaptiv	RBF adaptiv [14]	RBF $\epsilon = 10$	Chebyshev
$\|\cdot\|_\infty$	1.52e-07	3.52e-04	5.02e-01	9.24e-01

Tabell 4.3: *Sammenligning av nøyaktighet med $N = 198$ antall noder ($N = 194$ for RBF adaptiv [14].*

Tabell 4.4 viser resultatene fra den sammenlignebare adaptive metoden. Konvergensraten er sammenlignet med RBF adaptiv i figur 4.5. Metodene har det til felles at det tar noen iterasjoner før approksimasjonsfeilen minker nevneverdig. Merk at for RBF adaptiv fjernes det i siste iterasjon sentre uten at det legges til nye. Likevel fører dette til høyere nøyaktighet. Dette skjer ikke med vanlige kollokasjonsmetoder hvor man forventer at feilen synker når antall kollokasjonspunkt øker. For radielle basisfunksjoner spiller formparameteren like stor rolle som antall sentre.

Iteration	N	l_∞	l_2
0	34	0.968643	0.559801
1	45	0.912644	0.533539
2	74	0.172233	0.084677
3	98	0.415363	0.249337
4	158	0.014681	0.008472
5	172	0.001712	0.000305
6	194	0.000352	0.000211

Figur 4.4: *Tabellen viser konvergensen til den sammenlignbare adaptive RBF-metoden. Utklipp fra [14]*



Figur 4.5: *Plot av konvergensraten til RBF adaptiv og den sammenlignbare metoden [14]*

RBF adaptiv med linjemetoden

Diffusjonsligningen

Om vi tar poissonligningen (4.2) og erstatter $f(x)$ med $u_t(x,t)$ får vi diffusjonsligningen

$$u_t(x,t) = u_{xx}(x,t), \quad x \in (-1,1) \tag{4.7}$$

Dette er en partiell differensialligning i én romlig dimensjon, og utgjør et første forsøk på å teste RBF adaptiv med linjemetoden.

Randkravene settes til $u(-1,t) = u(1,t) = 0, \quad t > 0$ og følgende funksjon definerer initialverdiene ved $t = 0$:

$$f(x) = \begin{cases} x+1 & , x < 0 \\ 1-x & , x \geq 0 \end{cases}$$

Disse initialbetingelsene gir diskontinuitet i den førstederiverte av løsningsfunksjonen u. Her er funksjonen vanskeligere å approksimere, og vi forventer dermed en stor sentertetthet omkring $x = 0$. Ettersom t øker går glattes kurven og behovet for mange sentre vil avta.

Den analytiske løsningen finnes med separasjon av variable og fourierutvikling av initialverdiene [3]:

$$u(x,t) = \sum_{n=1}^{\infty} b_n \sin\frac{n\pi(x+1)}{2} \exp -\frac{(n\pi)^2 t}{4} \tag{4.8}$$

hvor

$$b_n = \begin{cases} 8/(n\pi)^2 & , n = 1,5,9,... \\ -8/(n\pi)^2 & , n = 3,7,11,... \\ 0 & , n = 2,4,6,... \end{cases}$$

I utregningen av (4.8) brukes den trunkerte funksjonen

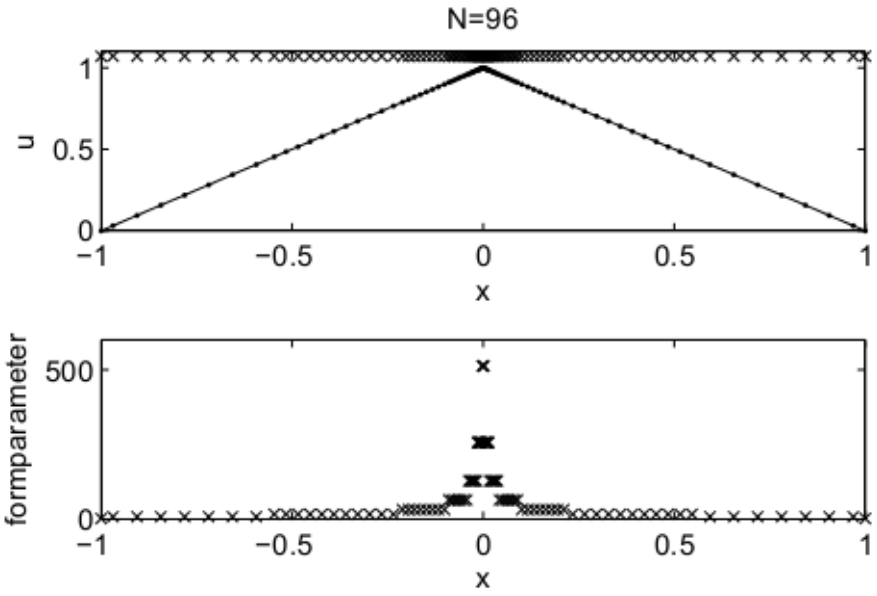
$$u_N(x,t) = \sum_{n=1}^{N} b_n \sin\frac{n\pi(x+1)}{2} \exp -\frac{(n\pi)^2 t}{4} \tag{4.9}$$

med N = 10^5. Trunkeringsfeilen ved fourierapproksimasjonen av $f(x)$ blir da $\|u_{10^5}(x,0) - f(x)\|_\infty$ = 3,69e-09 når normen blir evaluert på 200 uniformt fordelte punkt i $x = [-1,1]$.

Den adaptive metoden blir initiert med $N = 5$ sentre, alle med formparameter $\epsilon = 1$. Indikatorgrensen for å legge til nye sentre blir satt til *ovre*=1e-5 mens grensen for å fjerne eksisterende sentre blir satt til *nedre*=1e-10.

Tidsstegene blir gjort med *ode15s* med readaptering på hvert tidsintervall $\Delta t = 0.01$.

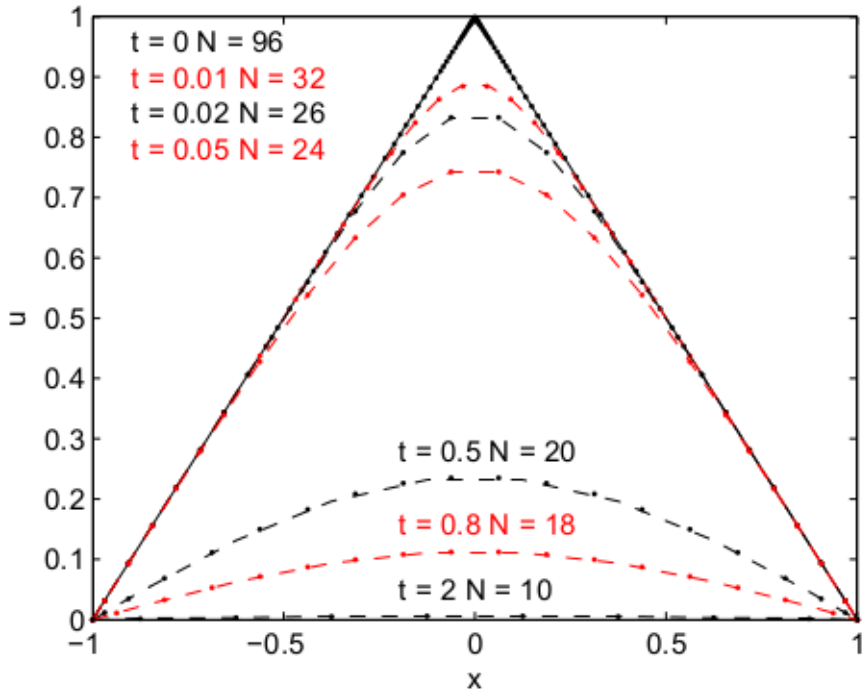
Figur 4.6 viser den approksimerte løsningen for $t = 0$ med senter- og formparameterfordeling etter første readaptering. Dette skjer i forkant av første tidssteg, og i praksis er dette interpolasjon av initialverdifunksjonen $f(x)$. For å ta en liten digresjon approksimerer RBF adaptiv $f(x)$ med feilen $\|u_{96}^{RBF} - f\|_\infty$ = 1,26e-05. For å oppnå en tilsvarende nøyaktighet for fourierutviklingen av $f(x)$ behøves omtrent 1000 ledd (da er $\|u_{1000} - f\|_\infty$ = 6,82e-05).



Figur 4.6: *Plot av approksimert løsning med senterfordeling (markert med 'x'). Plot av formparametrene (nedre figur).*

Allerede etter ett tidssteg er antall sentre redusert med en tredel til $N = 32$. Løsningsprofilen er da glattet ut, og dette gjenspeiles ved en lavere sentertetthet med lavere formparametre omkring $x = 0$. Figur 4.7 viser at antall sentre minker etterhvert som tiden øker og løsningsprofilen blir glattere og med stadig mindre variasjon.

Figur 4.8 viser hvordan approksimasjonsfeilen varierer som funksjon av tid for RBF adaptiv og for Chebyshev. RBF adaptiv skiller seg fra Chebyshev ved at feilen varierer mindre i tid. Kontrasten er spesielt stor ved de første tidsstegene. Her holdes variasjonen i feilen til RBF adaptiv lav ved å justere antall sentre. Metoder uten adaptivitet har ikke mulighet til å korrigere for at løsningsprofilen endrer seg fra skarp til glatt og vil da typisk ha minkende feil for økende tid. Merk at figur 4.8 også viser antall sentre for RBF adaptiv



Figur 4.7

i gitte tidsintervall. Ser man på RBF adaptiv ved en gitt t er approksimasjonsfeilen lavere enn for Chebyshev *med samme antall sentre*.

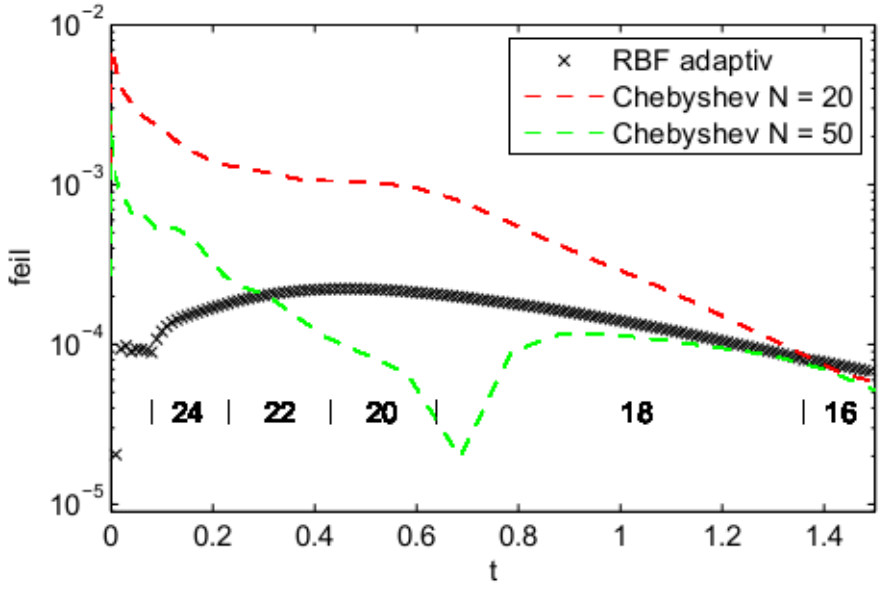
Tabell 4.4 viser kondisjonstallene til RBF adaptiv og Chebyshev derivasjonsmatrisene *uten randkrav*. Chebyshev derivasjonsmatrisen er tilnærmet singulær. Intuitivt er dette som forventet ettersom man ikke kan forvente en entydig løsning på et differensialproblem uten randbetingelser.

N	10	24	32	96
$\kappa(\boldsymbol{D}_{RBF})$	8,15e+03	9,94e+04	1,15e+05	1,40e+08
$\kappa(\boldsymbol{D}_{Cheb})$	4,89e+17	1.02e+17	2,12e+22	8,37e+17

Tabell 4.4: *Kondisjonstall til RBF adaptiv og Chebyshev derivasjonsmatriser.*

Kondisjonstallet til RBF adaptiv gjenspeiler derimot ikke dette, og har et lavt kondisjonstall. Denne egenskapen omtales i litteraturen som en 'regularization capability' [7], en egenskap som brukes for å approksimere 'ill-posed' problemer.

Tidsstegene gjøres med Matlabs *ode15s* som er en implisitt tidsstegmetode. For å undersøke om RBF adaptiv kan være velegnet for en eksplisitt metode,



Figur 4.8: *Plot av approksimasjonsfeilen til RBF adaptiv of Chebyshev. Tallene i figuren viser antall sentre til RBF adaptiv i tidsintervallet (merket '|')*

ser vi på egenverdiene til derivasjonsmatrisen. Derivasjonsmatrisen vi får ved å diskretisere (4.7) i rom er tilsvarende den som oppnås ved å diskretisere egenverdiproblemet

$$u_{xx} = \lambda u, \quad u(-1) = u(1) = 0 \tag{4.10}$$

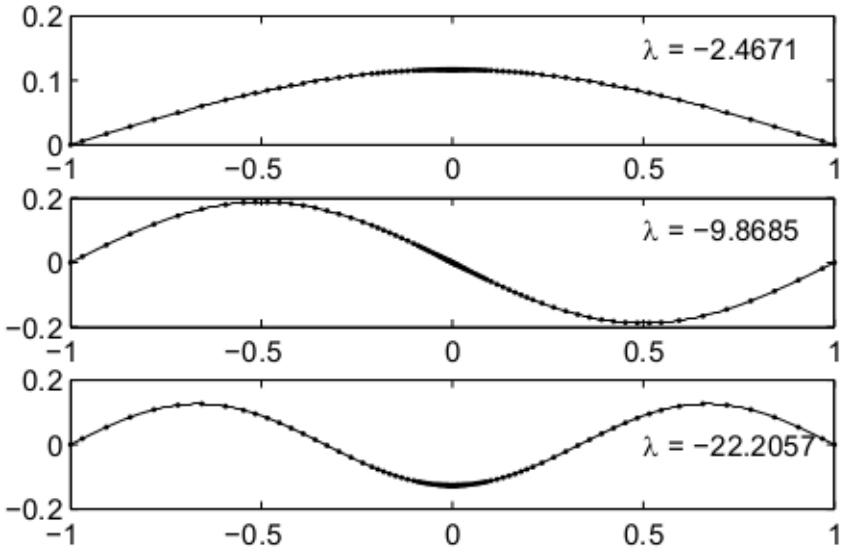
Dette problemet har eksakte egenverdier [17] $\lambda_n = -\pi^2 n^2/4, \quad n = 1, 2, ...$ med tilhørende egenfunksjoner $\sin(n\pi(x+1)/2)$.

Egenverdiproblemet (4.10) diskretiseres med senter- og formparameterfordelingen som ble generert av RBF adaptiv på (4.7). Merk at denne fordelingen er ikke ideell til (4.10) hvor egenfunksjonene har minkende bølgelengde for minkende λ.

Figur 4.9 viser egenverdiene og egenfunksjonene til RBF adaptiv derivasjonsmatrisen ved $t = 0$ ($N = 96$). Alle egenverdiene er reelle og negative. Den største egenverdien $\lambda = -2,47$ har en approksimasjonsfeil på 2,56e-04.

Stabilitetsområdet til eksplisitte metoder ligger nær origo, men på venstresiden av den imaginære aksen som vist i kapittel 1. Den minste reelle egenverdien må dermed skaleres med steglengden slik at alle egenverdiene befinner seg i stabilitetsområdet.

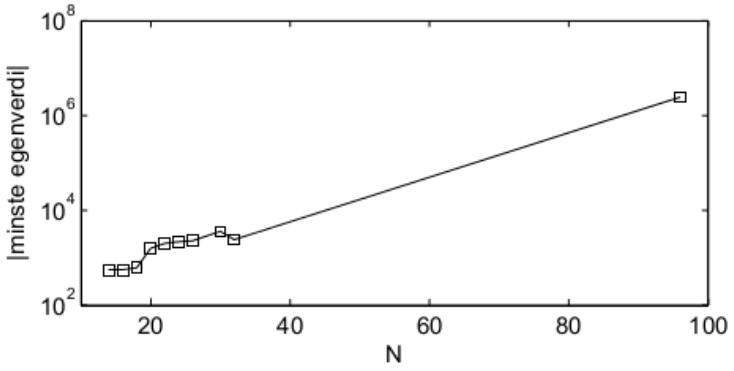
Figur (4.10) viser tallverdien av den minste egenverdien til diskretiseringenmatrisene som blir generert av RBF adaptiv på (4.7). Den minste egenverdien



Figur 4.9: *Plot av de tre første egenfunksjonene til RBF adaptiv derivasjonsmatrisen.*

ved $t = 0$ ($N = 96$) er av orden $\mathcal{O}(10^6)$, men allerede ved $t = 0,01$ ($N = 32$) er den $\mathcal{O}(10^3)$.

Med en eksplisitt tidsstegmetode kunne denne informasjonen vært brukt til å adaptivt justere steglengden. For at den minste reelle egenverdien skal befinne seg i metodens stabilitetsområde, er det nødvendig med små tidssteg for liten t hvor løsningsfunksjonen er skarp som gjør at N er stor. Deretter kan steglengden gradvis økes ettersom løsningsfunksjonen blir glattere, N minker og den minste negative egenverdien øker.



Figur 4.10: *Tallverdien til den minste egenverdien til RBF adaptiv derivasjonsmatrisen ved forskjellige N. N minker for økende t.*

Burgers ligning

Om vi legger til et ikke-lineært adveksjonsledd i diffusjonsligningen (4.7) får vi Burgers ligning

$$u_t(x,t) = -uu_x(x,t) + \nu u_{xx}, \quad x \in (-1,1), \quad t > 0 \tag{4.11}$$

I tillegg til diffusjon får vi en bølgebevegelse fra venstre mot høyre. Bølgen får en stadig skarpere front med økende t. Etterhvert som fronten dannes ønsker vi høyere sentertetthet omkring fronten. I tillegg ser vi om sentertettheten vil 'følge' frontens bevegelse mot høyre.

Løsningen på et slikt problem er gitt ved [15]

$$u(x,t) = \frac{0,1e^a + 0,5e^b + e^c}{e^a + e^b + e^c} \tag{4.12}$$

hvor $a = -(x + 0,5 + 4,95t)/(20\nu)$, $b = -(x + 0,5 + 0,75t)/(4\nu)$ og $c = -(x + 0,625)/(2\nu)$.

Initialverdier u(x,0) og randverdiene u(-1,t) og u(1,t) til (4.11) spesifiseres ved hjelp av den eksakte løsningen.

RBF adaptiv blir initiert med $N = 11$ sentre med formparameteren $\epsilon = 7$. Øvre og nedre terskel for å legge til eller fjerne sentre blir satt til *ovre* =1e-04 og *nedre*=1e-9. Konstanten i burgers ligning settes til $\nu = 0,0035$.

Tabell 4.5 sammenligner nøyaktigheten til RBF adaptiv med en adaptiv RBF metode fra litteraturen [15], samt en standard RBF metode med konstant formparameter. RBF adaptiv [15] har lite til felles med hvordan senterfordelingen genereres i denne oppgaven. Den er tatt som referanse fordi den også er basert på multikvadratiske basisfunksjoner.

Tabellen viser at nøyaktigheten til de to adaptive metodene er av samme størrelsesorden. For RBF med konstant formparameter trengs omtrent dobbelt så mange sentre for å oppnå tilsvarende nøyaktighet med en standard RBF metode. Avviket mellom de adaptive metodene skyldes variasjoner i senter- og formparameterfordelingen på grunn av forskjellige indikatorfunksjoner. Parameterne til RBF adaptiv er justert for å oppnå omtrent likt antall sentre som for RBF adaptiv [15].

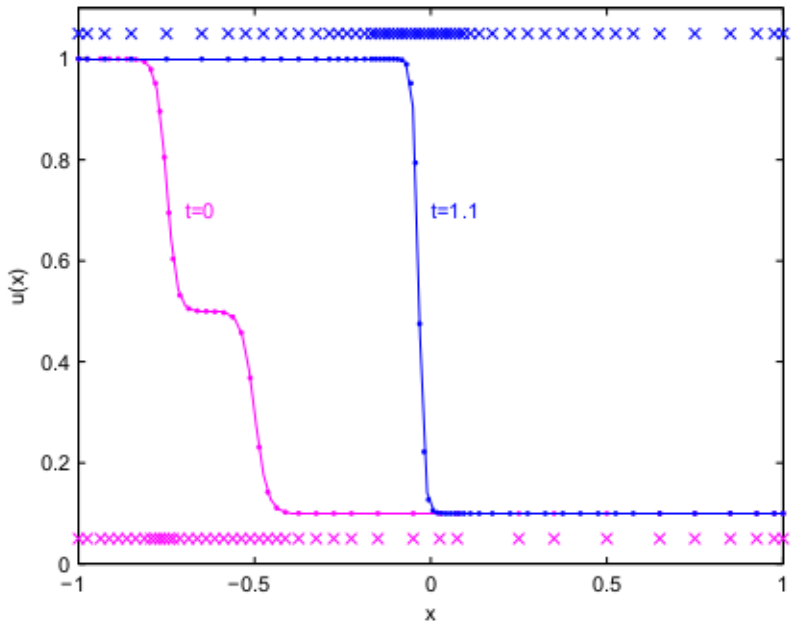
Figur 4.11 viser senterfordelingen til approksimasjonen av Burgers ligning ved tidene $t = 0$ og $t = 1.1$. Sentertettheten blir i økende grad konsentrert om fronten etterhvert som fronten blir stadig skarpere ved økende tid.

Metode	N	$\|\cdot\|_\infty$
RBF adaptiv	56	9,64e-03
RBF adaptiv [15]	47	1,91e-03
RBF $\epsilon = 11$	103	9,2e-03

Tabell 4.5: *Sammenligning av nøyaktighet for burgers ligning ved* $t = 1.1$.

Med Burgers ligning ser vi at RBF adaptiv også lykkes å omfordele sentrene hvor variasjonen i approksimasjonen forflyttes i domenet. Her 'følger' sentertettheten variasjonen etterhvert som den forflyttes i domenet, og flere sentre blir lagt til etterhvert som fronten i burgersligningen blir mer tydelig.

Matlabkoden som modellerer Burgers ligning og poissonproblemet er gjengitt i sin helhet i appendiks B.



Figur 4.11: *Figuren viser senterfordelingen til RBF adaptiv ved to tidspunkt. Burgers ligning.*

4.2 Resultater fra RBF adaptiv 2D

Poissonligningen med regulære render

Poissonligningen (4.1) i to dimensjoner blir

$$\frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} = f(x,y), \quad (x,y) \in \Omega \tag{4.13}$$

Modellproblemet lages med utgangspunkt i den kjente funksjonen

$$u(x,y) = e^{-\alpha(x^2+y^2)} \tag{4.14}$$

på domenet $[-1,1]^2$. Parameteren α bestemmer formen på løsningen. Høy α-verdi gir en funksjon med høy variasjon lokalisert på et lite område, mens lav verdi gir en funksjon med mindre variasjon lokalisert på et større område. Vi bruker Dirichlet randkrav med utgangspunkt i den eksakte løsningen.

Ved å substituere (4.14) i poissonligningen (4.13) får vi følgende uttrykk for høyresiden f:

$$f(x,y) = 4\alpha \ \mathrm{e}^{-\alpha\left(x^2+y^2\right)} \ \Big(\alpha(x^2+y^2) - 1\Big)$$

RBF adaptiv initieres med parametrene $N = 165$ (121 indre sentre, 44 randsentre). Formparameteren velges til $\epsilon = 2$. Øvre og nedre terskel for residualverdiene velges til *ovre* =1e-03 og *nedre*=1e-06. Parameteren α i den kjente løsningen settes til $\alpha = 10$.

Den adaptive metoden konvergerer etter kun to iterasjoner. Resultatene er vist i tabell 4.6.

Iterasjon	N	$\|\cdot\|_\infty$	kondisjonstall
0	165	2,94e-04	2,35e+07
1	253	5,54e-06	1,29e+08

Tabell 4.6: *Resultater fra poissonligningen i 2D med regulære render.*

Tabell 4.7 sammenligner resultatene fra RBF adaptiv med kollokasjon med konstant formparameter, Chebyshev basisfunksjoner samt RBF adaptiv fra litteraturen [6]. RBF adaptiv er implementert på grunnlag av beskrivelsen i [6]. Likevel gir RBF adaptiv i denne oppgaven noe bedre resultater. Approksimasjonsfeilen er bedre for identisk modellproblem. I tillegg konvergerer

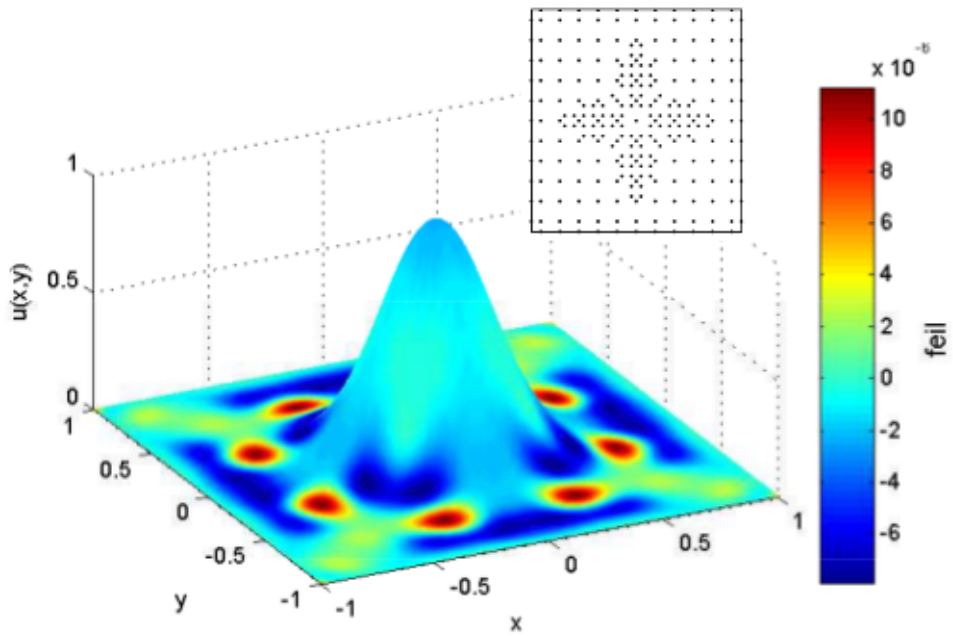
RBF adaptiv på to iterasjoner, mens RBF adaptiv [6] konvergerer på seks iterasjoner. Forskjellen kan trolig forklarer i hvordan metodene initieres. Det bør også nevnes at [6] fokuserer på beskrivelse av metoden, ikke prestasjon.

Nøyaktigheten til RBf adaptiv er omtrent lik som for kollokasjon med konstant formparameter, derimot er RBF adaptiv noe bedre kondisjonert. Merk at alle metodene med radielle basisfunksjoner oppnår høyere nøyaktighet enn Chebyshev med omtrent likt antall kollokasjonspunkt.

	RBF adaptiv	RBF adaptiv [6]	RBF $\epsilon = 1,5$	Chebyshev
$\|\cdot\|_\infty$	5,54e-06	4,54e-05	4,04e-06	4,89e-04
N	253	260	256	256
$\kappa(\boldsymbol{L}_N)$	1,30e+08	-	3,78e+10	1,30e+04

Tabell 4.7: *RBF adaptiv sammenlignet med tre andre kollokasjonsmetoder.*

Figur 4.12 viser løsningsfunksjonen, feilen, samt senterfordelingen til RBF adaptiv.



Figur 4.12: *Figuren viser løsningsfunksjonen til poissonligningen i to dimensjoner. Fargeskalaen viser fordelingen av approksimasjonsfeilen. Figuren i øvre hjørne viser senterfordelingen.*

Poisson med irregulære render

Simulering av fysiske prosesser fra virkeligheten skjer gjerne på domener som matematisk beskrives som irregulære. Det vil si at de er av en slik geometri som gjør matematikken vanskelig eller umulig. For poissonproblemer kan man bevise at det finnes en entydig løsning om visse vilkår for randen er tilstede, men den eksakte løsningen må approksimeres numerisk.

RBF adaptiv forsøkes på poissonproblemet (4.13) hvor domenet defineres på en entydig måte med parametrisert spline av en vilkårlig punktmengde. Punktmengdene som brukes i dette forsøket er fra tabell 4.8.

Tabell 4.8: *Punktmengdene som definerer C^2 rendene*

Ytre rand	$(x, y) = (-0.7, -0.8), (0, -0.8), (0.8, -0.8), (0.8, -0.3)...$ $(0, -0.3), (0, 0.3), (0.8, 0.3), (0.8, 0.8), (0, 0.8), (-0.7, 0.8)$
Indre rand	$(x, y) = (-0.5, -0.4), (-0.3, -0.4), (-0.4, 0.4)$

På den indre og ytre randen benyttes homogene Dirichlet randbetingelser

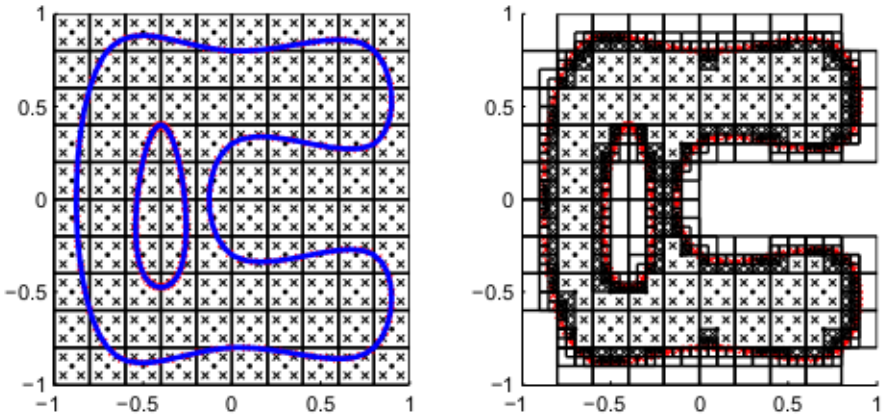
$$u(x, y) = 0, \quad (x, y) \in \partial\Omega$$

Høyresiden, f i (4.13) settes til $f = -10$ på hele domenet. Fysisk kan dette for eksempel tilsvare et konstant trykk på en membran fastmontert på rendene. Løsningen u vil da være høydeprofilen til membranen.

For dette problemet finnes det ingen eksakt løsning. Derfor bruker vi en approksimasjon gjort med MATLAB PDE Toolbox for å kunne gi RBF adaptiv et sammenligningsgrunnlag. Ved å velge en svært høy oppløsning på trianguleringen antas det at approksimasjonen fungerer som en tilnærmet eksakt løsning.

Før den adaptive prosessen kan begynne må boksene som i utgangspunktet utgjør et rektangulært nett tilpasses rendene. Dette er den geometriske forfiningsprosessen hvor bokser blir fjernet eller forfinet for å tilpasses det nye domenet definert av rendene.

Figur 4.13 viser boksene som utgjør domenet før og etter en slik prosess. Som forventet skjer det en opphopning av sentre nær rendene. Det er mulig å inkorporere den geometriske forfiningsprosessen i den adaptive prosessen, men siden feilen i RBF kollokasjonsmetoder uansett har tendens til å være størst nær rendene vil man trolig uansett ende opp med en senteropphopning slik figuren viser.



Figur 4.13: *Bokser før og etter geometrisk forfining.*

Etter den geometriske forfiningen initieres RBF adaptiv med formparameteren $\epsilon = 10$. Øvre og nedre terskel settes til *ovre* =1e-02 og *nedre*=1e-5.

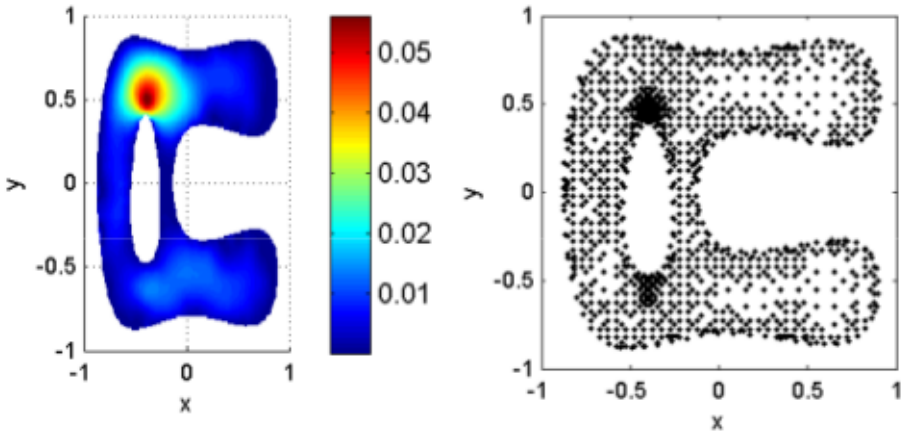
Tabell 4.9 vise resultatene fra iterasjonen. Metoden konvergerer etter seks iterasjoner. Da er feilen på samme størrelsesorden som ble satt for den øvre residualterskelen.

Tabell 4.9: *Resultater fra Poisson 2D med irregulære render*

#	N	$\|\cdot\|_\infty$	kondisjonstall
0	345	1,87e-01	2,98e+08
1	657	8,25e-02	8,55e+08
2	930	7,38e-02	2,60e+09
3	1095	6,64e-02	5,41e+09
4	1160	6,82e-02	1,88e+10
5	1203	5,74e-02	2,80e+10
6	1215	5,60e-02	3,67e+10

I figur 4.14 vises resultatene og 'feilen' sammenlignet med verdiene fra endelig elementapproksimasjonen. Punktverdiene er fra et 200 x 200 uniformt grid hvor både verdiene fra RBF adaptiv og PDE Toolbox er funnet ved lineær interpolasjon. Oppløftende er det at det virker som metoden lykkes med å øke nodetettheten i området hvor feilen er størst. Figur 4.14 viser en opphopning av sentre nord for den indre randen. Feilplottet er ganske karakteriserende for RBF kollokasjon hvor feilen har en tendens til å hope seg opp i et område i motsetning til en jevnere fordeling. Merk at den indre randen er asymmetrisk.

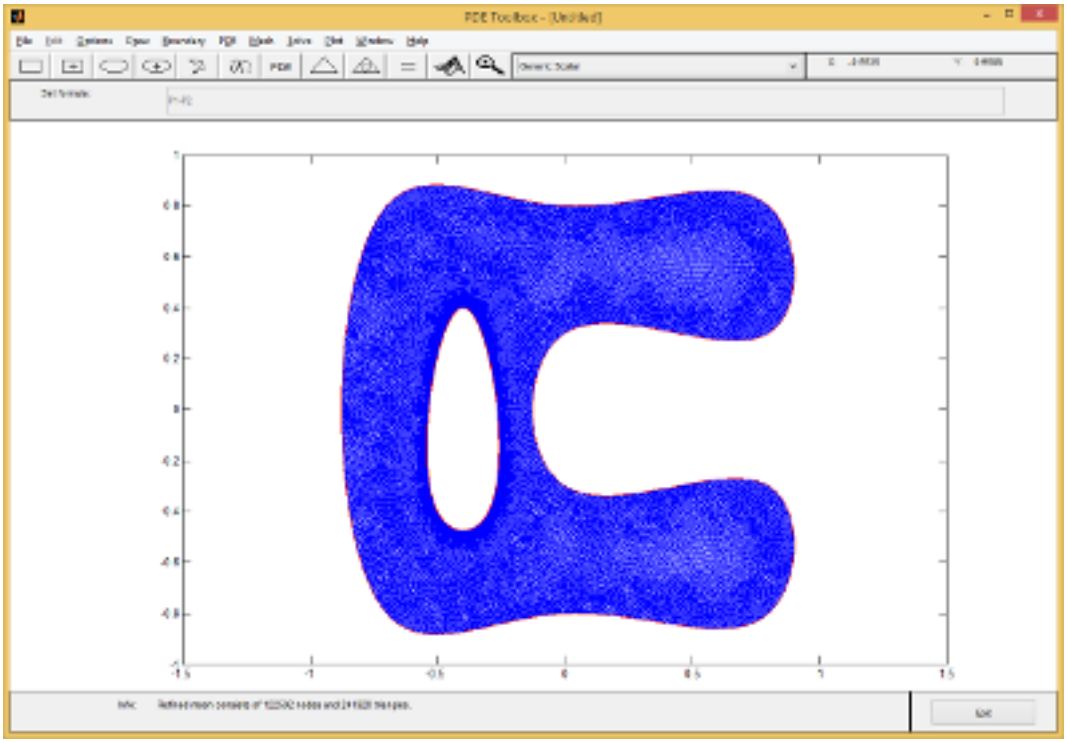
Figur 4.15 viser det triangulerte domenet bestående av 241920 triangler og



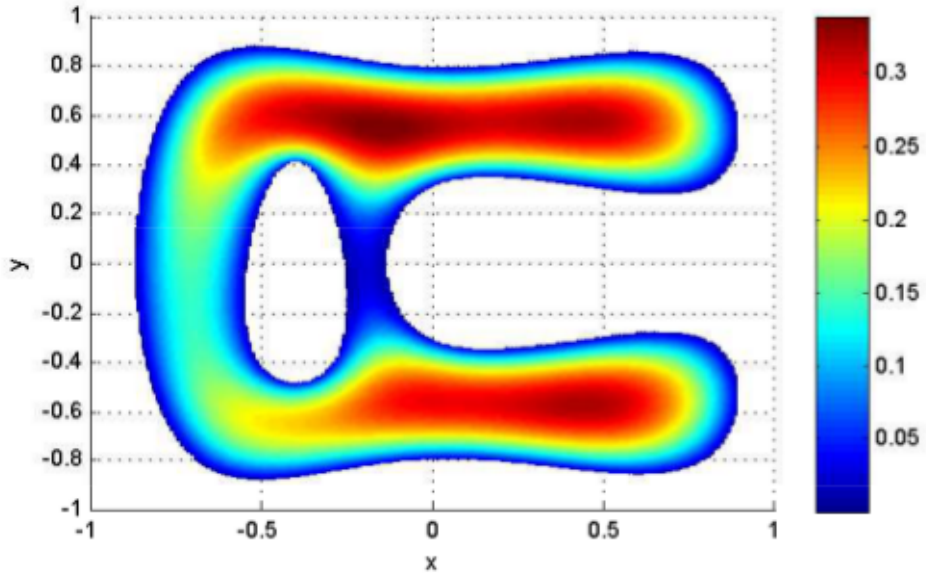
Figur 4.14: *Opphopning av sentre nord for den indre randen. Fargeskalaen viser approksimasjonsfeilen.*

122592 noder.

Figur 4.16 viser et plot av den approksimerte løsningen gjort med RBF adaptiv.



Figur 4.15: *Skjermbilde fra MATLAB PDE Toolbox.*



Figur 4.16: *Plot av funksjonsverdiene (fargeskala) til RBF adaptiv approksimasjonen*

Diffusjonsligningen

For å teste datastrukturen til den todimensjonale RBF adaptiv med linjemetoden implementeres den for diffusjonsligningen

$$u_t = u_{xx} + u_{yy} \tag{4.15}$$

på det rektangulære domenet $[-1, 1]^2$ med homogone Dirichlet randbetingelser.

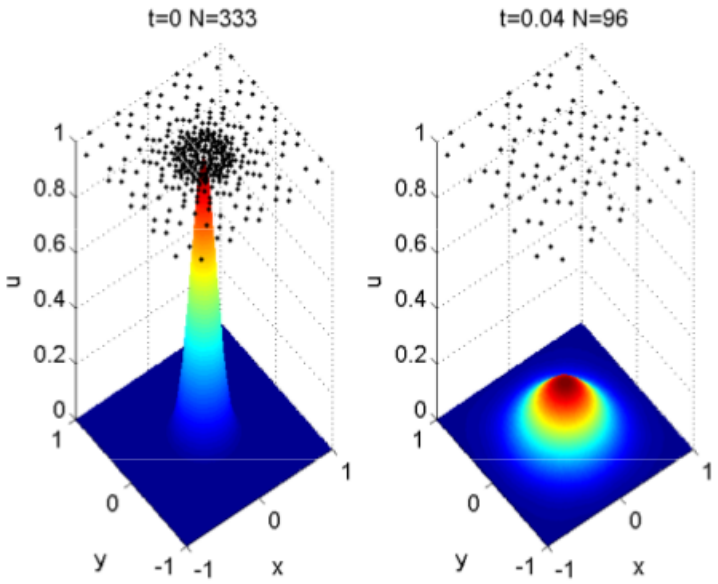
Som initialkrav ved $t = 0$ velges funksjonen

$$f(x, y) = T_{max} \exp - \frac{x^2 + y^2}{\sigma^2}$$

Parametrene settes i forsøket til $T_{max} = 1$ og $\sigma = 0.2$.

RBF adaptiv initieres med $N = 45$ sentre (25 indre sentre, 20 randsentre). Formparameteren settes til $\epsilon = 5$. Øvre og nedre terskel settes til henholdsvis $ovre$ =1e-03 og $nedre$ =1e-06. Griddet readapteres hvert tidsintervall Δt=1e-03

Figur 4.17 viser den approksimerte løsningen ved $t = 0$ og $t = 0.04$. Ved $t = 0$ konvergerer readapteringen med $N = 333$ sentre. Approksimasjonsfeilen er



Figur 4.17: *Figuren viser approksimasjonen og senterfordelingen ved tiden $t = 0$ og $t = 0.04$.*

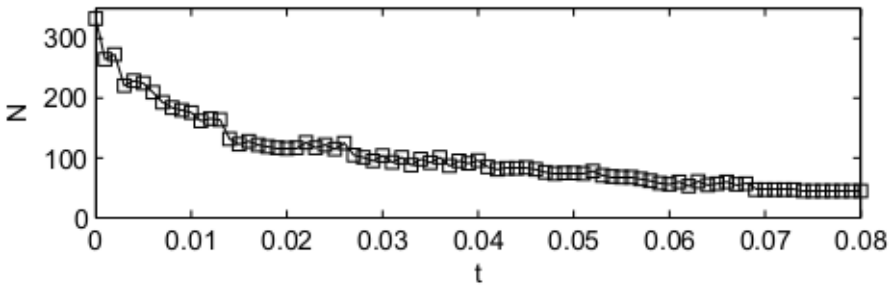
da $\|u_{333} - f\|_\infty = 9.65\text{e-}04$. Ved tiden $t = 0.04$ er antall sentre redusert til $N = 96$.

Kondisjonstallet og egenverdiene til derivasjonsmatrisen følger samme trend som for diffusjonsligningen i én dimensjon. Kondisjonstallene er lave (se tabell 4.10), noe som indikerer at startparametrene kan velges på en måte som gir bedre nøyaktighet.

Tabell 4.10: *Kondisjonstall for derivasjonsmatrisen*

N	333	180	95	58
$\kappa(\boldsymbol{D}_{RBF})$	2,34e+04	1,20e+04	2,08e+03	7,23e+02

Figur 4.18 viser et plot av antall sentre som funksjon av tid. Også her avtar antall sentre som forventet etterhvert som initialfunksjonen f glattes ut i diffusjonsprosessen.



Figur 4.18: *Figuren viser antall sentre N som funksjon av tiden t.*

Kapittel 5

Oppsummering og konklusjoner

En adaptiv kollokasjonsmetode har blitt implementert og undersøkt. Metodens senterfordeling ble basert på enkle antagelser underbygget av de numeriske forsøkene gjort i kapittel to.

Ved å fordele sentrene etter forhold ved problemet ble det vist at RBF adaptiv gir høyere nøyaktighet per antall sentre enn standard RBF- og Chebyshev kollokasjon.

Formparametrene ble fordelt etter forhold ved metoden. Om formparameteren holdes konstant vil kondisjonstallet til RBF kollokasjonsmatrisen øke når senteravstanden minker. Ved å bruke datastrukturen ble basisfunksjonene i områder med høy sentertetthet tildelt større formparameterverdi. Dermed ble kondisjonstallet holdt på et forsvarlig nå.

Det ble demonstrert at den adaptive metoden kan ha eksponensiell konvergensrate tilsvarende Chebyshev kollokasjonsmetoden. I tillegg har metoden en fleksibilitet med tanke på geometri som ikke finnes hos pseudospektralmetoder.

Den største ulempen med RBF adaptiv knyttes til kompleksitet i form av høyt antall regneoperasjoner. Den største teoretiske utgiftsposten er invertering av en full koeffisientmatrise. I praksis viste det seg at operasjoner på datastrukturen tok en betydelig andel av regnekraften. Årsaken ligger i måten datastrukturen ble implementert i Matlab. I teorien kan operasjoner på datastrukturen implementeres med en kostnad på $\mathcal{O}(\log N)$ for å legge til eller fjerne noder.

Selv om prinsippene bak senterfordelingen samsvarer med vår intuisjon, er det et par aspekt ved radielle basisfunksjoner som motstrider hva vi vanligvis

forventer fra en kollokasjonsmetode:

- For standard kollokasjonsmetoder forventes det at approksimasjonsfeilen vil synke etterhvert som antall kollokasjonspunkt øker. Slik er det ikke med radielle basisfunksjoner. Her er formen på basisfunksjonen, som bestemmes av formparameteren, vel så viktig. Justeres ikke formparameteren korrekt som funksjon av senteravstanden vil metoden divergere for økende N. Det ble vist i forsøket med 1D poissonproblemet at nøyaktigheten kan øke for *minkende* N.
- RBF diskretisering gir opphav til derivasjonsmatriser som kan være godt kondisjonert selv i tilfeller hvor det kontinuerlige problemet ikke har en entydig løsning. Dette ble oppdaget ved å plotte kondisjonstallene til derivasjonsmatrisen hvor randbetingelsene ikke er implementert. For standard kollokasjonsmetoder vil diskretisering av et ikke-korrekt stilt problem gjenspeiles i en singulær eller høyt kondisjonert derivasjonsmatrise.

Datastrukturene

Kjernen i implementeringen av den adaptive metoden, kd-tre datastrukturene, har vist seg å være vellykket av flere grunner:

- Datastrukturens rekursive natur gjør at antall sentre i et område kan økes eksponensielt. Dette kan være en lovende egenskap for problemer som beskriver fenomener som er lokalisert på et lite område i et stort domene. Et eksempel på dette er den tidsavhengige schrödingerligningen hvor en bølge forflytter seg i et domene som i prinsippet er av uendelig utstrekning.
- Radielle basisfunksjoner har stor fleksibilitet ovenfor irregulær geometri, og det ble vist at også kd-tre datastrukturer har fleksibilitet som skal til for å håndtere dette. Dette ble gjort ved en geometrisk forfiningsprosess i forkant av den adaptive iterasjonen.
- Ved å utnytte hierarkiet i datastrukturen tilpasses formparametrene på en enkel og naturlig måte. Kostnaden i form av regneoperasjoner ble sammenlignet med å kalkulere avstanden til nærmeste nabosenter er minimal.
- Implementert rett er dette en billig datastruktur med tanke på antall regneoperasjoner. Det er også en enkel datastruktur. At RBF gir opp-

hav til gitterfrie metoder gjør arbeidet med implementering av adaptivitet enklere enn for gitteravhengige metoder.

- Kostnaden med å legge til eller fjerne noder er kun avhengig av størrelsen på datatreet. I motsetning til trianguleringsalgoritmer er kostnaden uavhengig av dimensjonen på problemet.

Resultatene

Den adaptive metoden har blitt implementert både for én- og todimensjonale problemer, men hovedfokuset i arbeidet har vært på å forstå RBF adaptiv på modellproblemer i 1D.

For modellproblemer med høy variasjon på et lite intervall (poissonligningen med $u(x) = \tan(\eta x)$) viste RBF adaptiv seg overlegen sammenlignet med Chebyshev og RBF med konstant formparameter. Indikatorfunksjonen basert på residualen viste seg velegnet til å fange opp slike variasjoner.

For problemer med tidsledd ble det brukt en indikatorfunksjon basert på interpolasjon som ga egnet senterfordeling for to fysiske fenomener:

- For diffusjonsligningen ble diskontinuiteten i den deriverte av initialfunksjonen fanget opp og 'pakket inn' med sentre. Etterhvert som funksjonen ble glattet ut ble dette gjenspeilet med lavere tetthet i senterfordelingen.
- For Burgers ligning viste indikatorfunksjonen seg egnet til å fange opp den bevegelige fronten, og sentertettheteten 'flyttet seg' med den. Etterhvert som fronten ble skarpere ble sentertettheten økende.

For todimensjonale problemer har RBF adaptiv først og fremst blitt implementert av to grunner:

- Det ble vist at datastrukturen er lett generaliserbar til flerdimensjonale problemer. Her er kontrasten til Chebyshev kollokasjonsmatrisen stor. Hos Chebyshev settes derivasjonsmatrisen settes sammen med tensorprodukt, noe som medfører at antall kollokasjonspunkt vil øke eksponensielt med antall dimensjoner.
- Metoden implementeres i 2D for å demonstrere at datastrukturen har den fleksibiliteten som skal til for å håndtere irregulære render. Det ble også demonstrert at RBF adaptiv i 1D kan parametriseres for adaptivitet på randen. Ved denne tilnærmingen vises en rekursivitet i måten

RBF adaptiv generaliseres til k dimensjoner. Randen til RBF adaptiv i k dimensjoner kan alltid parametriseres med RBF adaptiv i k-1 dimensjoner.

For RBF adaptiv 2D ble den samme strategien som for 1D brukt ved at nye sentre ble tildelt dobbelt så stor formparameter som foreldersenteret i datastrukturen. For RBF adaptiv 1D fungerte denne strategien godt. I 2D er samspillet mellom sentrene langt mer komplisert, og oppgavens begrensede tidsramme har ikke tillatt en detaljert undersøkelse av hvordan formparameteren kan fininstilles som funksjon av senteravstanden.

Trolig kunne resultatene i 2D vært forbedret ved å ganske enkelt bruke mer tid på justering av parametrene som initierer metoden. For eksempel viser figur 2.11 i kapittel 2 at ved å endre formparameteren ϵ fra 0,65 til 1,00 oppnås en forbedret nøyaktighet med en faktor på omtrent 1e-4.

Videre arbeid

For realistiske simuleringer trengs langt flere kollokasjonspunkt enn det som har blitt brukt i denne oppgaven. Metoder for å redusere kompleksiteten til den adaptive metoden bør dermed være fokus i framtidig arbeid.

- Det har blitt kjent en diskretiseringsstrategi tilsvarende det som gjøres i endelig differansemetoden [1]. For et gitt senter blir det laget en lokal diskretiseringsstensil med utgangspunkt i et antall nærmeste sentre. For k-d tre datastrukturer finnes det algoritmer for å finne nærmeste sentre på en billig måte. Adaptivitet kan implementeres ved å lage høyere ordens stensiler i aktuelle områder. I tillegg kan det undersøkes om strukturen på den glisne diskretiseringsmatrisen kan brukes til å finne koeffisientvektoren på en billigere måte enn ved å invertere fulle matriser.
- Formparameteren spiller en avgjørende rolle for å holde kondisjonstallet på et tilfredsstillende nivå. Å definere formparameteren som funksjon av senteravstanden fungerte godt i 1D. For flerdimensjonale problemer blir interaksjonen mellom sentrene mer komplisert, og mer arbeid må legges i å finne gode formparameterfunksjoner.
- Forsøket med diffusjonsligningen viste at antallet sentre synker når initialfunksjonen glattes ut. Ved å bruke eksplisitte tidsstegmetoder kan kompleksiteten reduseres, og ved å finne en indikatorfunksjon for steglengden kan det også sørges for adaptivitet i tid.

Matematikk er først og fremst kjent som en deduktiv vitenskap, og skal radielle basisfunksjoner få en plass blant de 'Tre Store' behøves mer teori om hvor godt gitte senter- og formparameterfordelinger approksimerer løsningen. Eksperimentell numerisk matematikk kan ha rollen som veiviser, men resultatene bør følges opp av bekreftende teori.

Tillegg A

Basisfunksjoner brukt i oppgaven

Multikvadratiske basisfunksjoner	
$\phi(r) = \sqrt{(\epsilon r)^2 + 1}$	
1D	
$r = \|\boldsymbol{x}\| = \sqrt{x^2}$	
$\frac{d}{dx}\phi(r) =$	$\epsilon^2 x / \sqrt{(\epsilon r)^2 + 1}$
$\frac{d^2}{dx^2}\phi(r) =$	$\epsilon^2 / ((\epsilon r)^2 + 1)^{3/2}$
2D:	
$r = \|\boldsymbol{x}\| = \sqrt{x^2 + y^2}$	
$(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2})\phi(r) =$	$\epsilon^2((\epsilon r)^2 + 2)/((\epsilon r)^2 + 1)^{3/2}$

Tillegg B

Implementering i Matlab

Figur B.1 viser organiseringen av filene i vedlegget. Matlabfilen *burgers.m* produserer en figur som viser den approksimerte løsningen til poissonligningen med senterfordeling. Figuren blir oppdatert for hvert tidssteg. Tilsvarende produserer *poisson.m* figuren til 1D poissonligningen med senterfordeling.

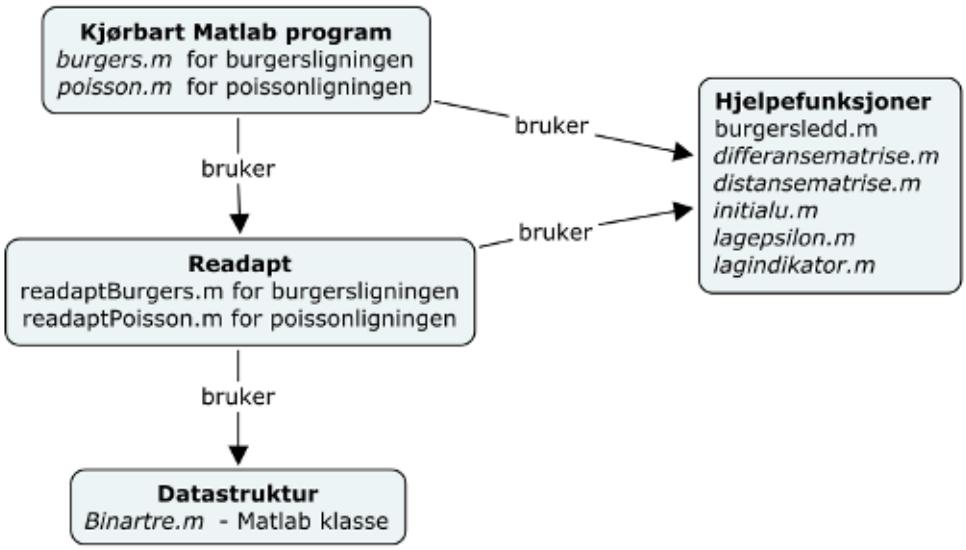
Koden er organisert i et antall filer. Å organisere koden etter hvilken funksjon den har gjør det lettere å holde oversikten. Dette er viktig når det eksperimenteres med flere typer modellproblemer. I tillegg unngås unødvendig duplisering av koden.

Koden for RBF adaptiv 2D er av plasshensyn ikke vedlagt. Forskjellene i koden er små, og største forskjellen ligger i datastrukturen. For RBF adaptiv 2D brukes en forfiningsstrategi hvor sjekkpunktene i en boks blir omgjort til nye sentre uavhengig av boksens andre sjekkpunkt. Dette fører til en litt mer komplisert datastruktur fordi det må holdes styr på orienteringen til en nodes barnnoder.

Implementering av datastrukturen

Rekursiver strukturer kan i prinsippet programmeres elegant og med et minimalt antall kodelinjer. En minimal datastruktur vil i prinsippet være mulig å oppnå med kun basisfunksjonens senter som datafelt.

Binærtreet er implementert for å inneholde data som uansett vil være nødvendig for å sette opp kollokasjonsmatrisen. Datastrukturen er implementert som en tabell hvor elementreferansene i tabellen brukes for å opprettholde



Figur B.1: *Diagram over vedlagte filer.*

den hierarkiske strukturen. Dette medfører noe 'bokføring' i forbindelse med å oppdatere elementreferansene når noder fjernes eller når nye noder legges til strukturen.

Tabell B.1 gir en oversikt over datafeltene i implementeringen av binærtreet.

Tabell B.1: *Oversikt over datafelt i hver node til binærtreet*

Datafelt	*Beskrivelse*
ID	Unik tabellreferanse til hver node i treet
Senter	Senterkoordinat (kollokasjonspunkt)
Child	Tabellreferanse til nodens barn
Parent	Tabellreferanse til nodens forelder
Nivaa	Nodens nivå
Dim	Intervallets lengde
Rand	Indikerer om intervallet er på randen av domenet
Sjekkpkt	Intervallets sjekkpunkter
Epsilon	Formparameter i senterpunktet
Indikator	Tallverdi for å avgjøre om sjekkpunktene skal forfines/fjernes

Matlab kode

poisson.m

```
% poisson.m - Kjørbart Matlab program
%
% Simulerer poissonligningen (4.2). Programmet produserer figur
% av u_N og senterfordelingen.


clear
close

% Runge testproblem 1D
u = @(x) 1./(25*x.^2+1); % eksakt løsning
f = @(x) (5000.*x.^2)./(25.*x.^2 + 1).^3 - 50./(25.*x.^2 + 1).^2;

% multikvadratisk basisfunksjoner
rbf = @(c,r) sqrt((c.*r).^2 + 1);
ddrbf = @(c,r) c.^2./(c.^2.*r.^2 + 1).^(3/2);

% ------- parametre
N = 3;
epsilon = 0.1;
ovre = 1e-7;
nedre = ovre*1e-9;

% ------- adaptiv prosess
[lambda, data, STAT] = ...
    readaptpoisson(N, epsilon, ovre, nedre, rbf, ddrbf, f, u);

% ------- plot løsning
EM = rbf(lagepsilon(length(data.eps),data.eps), ...
    distansematrise(data.sentre, data.sentre));
uN = EM*lambda;

subplot(2,1,1)
[ix,i] = sort(data.sentre);
plot(ix,uN(i))
feil = norm(uN-u(data.sentre),'inf');
title(['N = ' num2str(length(uN)) ', feil = ' num2str(feil)])
ylabel('u')
xlabel('x')

hold on
plot(data.sentre,0,'kx')
axis([-1,1,-0.1,1.1])
```

```
hold off

subplot(2,1,2)
plot(data.sentre,data.eps,'x')
ylabel('formparameter')
xlabel('x')
```

readaptpoisson.m

```
% readaptpoisson.m - Funksjon.
%
% Implementering av den adaptive prosessen. Legger til eller fjerner
% noder fra binærtreet.
function [lambda, data, STAT] = ...
    readaptpoisson(N, epsilon, ovre, nedre, rbf, ddrbf, f, u)

tre = Binartre;
tre = initier(tre,N,epsilon); %Initielt grid med bokser

stopp = 0;
iterasjon = 1;
while stopp == 0
senteraktiver = 0;

STATnyboks = 0;
STATfjernsenter = 0;

STAT{1}.N = N;
STAT{1}.epsilon = epsilon;
STAT{1}.ovre = ovre;
STAT{1}.nedre = nedre;


data = hentdataA(tre);

% finner lambda
LN =[ddrbf(lagepsilon(size(data.indresentre,1),data.eps), ...
    distansematrise(data.indresentre,data.sentre));
    rbf(lagepsilon(size(data.randsentre,1),data.eps),...
    distansematrise(data.randsentre,data.sentre))];

rhs = [f(data.indresentre(:,1)); u(data.randsentre(:,1))];
lambda = LN\rhs;

% ------ lag indikator
indikator = lagindikator(lambda, f, data, ddrbf);
```

```
% ------- last inn indikatorverdiene

for i = 1 : length(tre)
end

ID = data.ID;
for i = 1 : length(ID)
    areal = tre(ID(i)).Dim;

    tre(ID(i)).Indikator = [tre(ID(i)).Indikator; areal*indikator(i)];
end

forfine = [];
fjerne = [];

for i = 1 : length(tre)
if tre(i).Skjulsenter == 0
% ------- sjekker om indre intervall skal forfines eller fjernes
% hvis boksen ikke er på randen og har to sjekkpunkt
if tre(i).Rand == 0 && size(tre(i).Indikator,1) == 2
    % fjernes hvis begge sjekkpunktene er under
    % fjernesterskelen og er ikke på nivaa 0
    if sum(tre(i).Indikator < nedre*ones(2,1)) == 2 && tre(i).Nivaa ~= 0
        fjerne = [fjerne; i];
    end
    % forfines hvis begge sjekkpunktene er over forfinesterskelen
    if sum(tre(i).Indikator >= ovre*ones(2,1)) == 2
        forfine = [forfine; i];
    end
end
% Sjekk om randintervallet skal forfines
if tre(i).Rand == 1
    if tre(i).Indikator >= ovre == 1
        forfine = [forfine;i];
    end
end
end
tre(i).Indikator = []; % nullstiller til neste iterasjon
end

STAT{iterasjon+1}.tre = tre;
STAT{iterasjon+1}.indikator = indikator;
STAT{iterasjon+1}.lambda = lambda;
STAT{iterasjon+1}.kond = cond(LN);


% ------ forfiner bokser
for i = 1 : length(forfine)
```

```
    if tre(forfine(i)).Rand == 0
        tre = nyboks(tre,forfine(i),1);
        tre = nyboks(tre,forfine(i),2);
        tre(forfine(i)).Skjulsenter = 1; % skjuler senteret
        STATnyboks = STATnyboks + 2;
        STATfjernsenter = STATfjernsenter + 1;
    else
        tre = nyboks(tre,forfine(i),1);
        STATnyboks = STATnyboks + 1;
        % oppdaterer epsilonverdien til randpunktet
        tre(forfine(i)).Epsilon = tre(end).Epsilon;
    end
end

% ------- fjerner bokser
fjerne = sort(fjerne,'descend');
for i = 1 : length(fjerne)
       [tre, aktiver] = fjernboks(tre,fjerne(i));
        senteraktiver = senteraktiver + aktiver;
       STATfjernsenter = STATfjernsenter + 1;
end

STAT{iterasjon+1}.antfjernes = STATfjernsenter;
STAT{iterasjon+1}.antnye = STATnyboks;
STAT{iterasjon+1}.senteraktiveres = senteraktiver;

if STATnyboks == 0 && STATfjernsenter == 0
    stopp = 1;
    siste = 0;
elseif iterasjon > 9
    stopp = 1;
    siste = 1;
end

disp(['------ Iterasjon ' num2str(iterasjon) '-------'])
disp(['Nye sentre: ', num2str(STATnyboks)])
disp(['Fjerner sentre: ', num2str(STATfjernsenter)])
iterasjon = iterasjon + 1;

end

% oppdater lambda og data etter siste iterasjon
if siste == 1
    data = hentdataA(tre);
    LN =[ddrbf(lagepsilon(size(data.indresentre,1),data.eps), ...
        distansematrise(data.indresentre,data.sentre));
        rbf(lagepsilon(size(data.randsentre,1),data.eps),...
        distansematrise(data.randsentre,data.sentre))];
```

```
    rhs = [f(data.indresentre(:,1)); u(data.randsentre(:,1))];
    lambda = LN\rhs;

    STAT{iterasjon+1}.tre = tre;
    STAT{iterasjon+1}.indikator = indikator;
    STAT{iterasjon+1}.lambda = lambda;
    STAT{iterasjon+1}.kond = cond(LN);
    STAT{iterasjon+1}.antfjernes = STATfjernsenter;
    STAT{iterasjon+1}.antnye = STATnyboks;
end

end
```

burgers.m

```
% burgers.m - Kjørbart Matlab program
%
% Simulerer burgers ligning (4.10)
% Programmet produserer figur av u_N med senterfordeling.
% Figuren oppdateres for hvert tidssteg.

clear
close

% ------------ multikvadratisk basisfunksjoner
rbf = @(c,r) sqrt((c.*r).^2 + 1);
drbf = @(c,r,dx) (c.^2.*dx)./(r.^2.*c.^2 + 1).^(1/2);
ddrbf = @(c,r) c.^2./(c.^2.*r.^2 + 1).^(3/2);

% ------------ parametre
startepsilon = 7; % formparameter
startN = 11;      % antall sentre før forfining
ovre = 1e-4;      % terskel for nye sentre
nedre = 1e-9;     % terskel for å fjerne
tid = 0;
plotgap = 1;
tmax = 1.1;
dt = 0.01;

% ---------- initier Kvadtre datastruktur
tre = Binartre;
tre = initier(tre,startN,startepsilon);

% ---------- readapt før tidssteg
[sentre, uN, Dx, Dxx, data] = ...
    readaptburgers(tre, ovre, nedre, rbf, drbf, ddrbf);
```

```
% ---------- plot figur før tidssteg
[x, I] = sort(sentre);
y = uN(I);
px = linspace(-1,1)';
datau = interp1(x,y,px);
plot(px, datau,'k')
title(['N = ' num2str(length(x)) ' t = ' num2str(tid)])
axis([-1,1,0,1.1])
hold on
plot(x,0.05,'kx')
plot(x,y,'k.')
hold off
drawnow

% ---------- start tidsstegene
antalldt = round(tmax/dt);
for kk = 1 : antalldt
disp([' --------- Iterasjon ' num2str(kk), '----------'])

 % Dirichlet homogene randkrav
b = find(sentre==1);
u1 = initialu(sentre(b),0);
c = find(sentre == -1);
u2 = initialu(sentre(c),0);

Masse = eye(length(uN)); Masse(b,b) = 0;
options = ...
    odeset('Mass',Masse,'MstateDependence','no','MassSingular','yes');
f = @(t,u) burgersledd(t,u,Dx,Dxx,u1,b,u2,c);

[T,U] = ode15s(f,[0, dt],uN,options);
tid = tid + T(end);
uN = U(end,:);

% ---------- readapt
tre = Binartre;
tre = initier(tre,startN,startepsilon);
[sentre, uN, Dx, Dxx, data] = ...
    readaptburgers(tre, sentre, uN, ovre, nedre, rbf, drbf,ddrbf);

% ---------- plot figur
if mod(kk, plotgap) == 0
    [x, I] = sort(sentre);
    y = uN(I);
    px = linspace(-1,1)';
    datau = interp1(x,y,px);

    plot(px, datau,'k')
```

```
    title(['N = ' num2str(length(x)) ' t = ' num2str(tid)])
    axis([-1,1,0,1.1])
    hold on
    plot(x,0.05,'kx')
    plot(x,y,'k.')
    hold off
    drawnow
    end
end
```

readaptburgers.m

```
% readaptburgers.m - Funksjon.
%
% Implementering av den adaptive prosessen. Legger til eller fjerner
% noder fra binærtreet.
function [sentre, uN, Dx, Dxx, data] = readaptburgers(varargin)
if length(varargin) == 6
    tre = varargin{1};
    ovre = varargin{2};
    nedre = varargin{3};
    rbf = varargin{4};
    drbf = varargin{5};
    ddrbf = varargin{6};
    start = 1;
elseif length(varargin) == 8
    tre = varargin{1};
    uNsentre = varargin{2};
    uN = varargin{3};
    ovre = varargin{4};
    nedre = varargin{5};
    rbf = varargin{6};
    drbf = varargin{7};
    ddrbf = varargin{8};
    start = 0;
end

antit = 6;
stopp = 0;
teller = 0;
%for k = 1 : antit
while stopp == 0 && teller < 10
data = hentdataA(tre); % henter data fra Binartre datastruktur
sentrefor = data.sentre;

% ----- Setter sammen derivasjonsmatrisen Dx
KM = drbf(lagepsilon(length(data.eps),data.eps), ...
```

```
    distansematrise(data.sentre,data.sentre),...
    differansematrise(data.sentre,data.sentre));

A = rbf(lagepsilon(length(data.eps),data.eps), ...
    distansematrise(data.sentre,data.sentre));

Dx = KM/A;

% ----- Setter sammen derivasjonsmatrisen Dxx
KM = ddrbf(lagepsilon(length(data.eps),data.eps), ...
    distansematrise(data.sentre,data.sentre));

A = rbf(lagepsilon(length(data.eps),data.eps), ...
    distansematrise(data.sentre,data.sentre));
Dxx = KM/A;

% ----- Setter sammen evalueringsmatrisen
EM = rbf(lagepsilon(length(data.sjekkpkt),data.eps), ...
    distansematrise(data.sjekkpkt,data.sentre));

% ----- henter initialverdiene hvis tidsstegene ikke er startet
if start == 1
    uN = initialu(data.sentre,0);
    uNsentre = data.sentre;
end

% ----- lager prediktor
[sentresortert, I] = sort(uNsentre);
uNsortert = uN(I);
f = @(x) interp1(sentresortert,uNsortert,x,'spline');
lambda = A\f(data.sentre);
prediktor = abs(EM*lambda - f(data.sjekkpkt));

% STAT
STATs0 = length(data.sentre);

forfine = [];
fjerne = [];

% ----- prosessen hvor intervall fjernes eller legges til
for i = 1 : length(tre)
if tre(i).Skjulsenter == 0
    % nullstill prediktor fra forrige iterasjon
    tre(i).Prediktor = [];
    % laster inn nye prediktorverdier
    tre(i).Prediktor = prediktor(1:size(tre(i).Sjekkpkt,1));
    prediktor(1:size(tre(i).Sjekkpkt,1)) = [];

    % ------- Finner intervall som forfines/fjernes
```

```
    if tre(i).Rand == 0 && size(tre(i).Prediktor,1) == 2
        % intervallet fjernes hvis begge sjekkpunktene er under
        % sterskelen og er ikke på nivaa 0
        if sum(tre(i).Prediktor < nedre*ones(2,1)) == 2 && tre(i).Nivaa ~= 0
            fjerne = [fjerne; i];
        end
        % forfines hvis begge sjekkpunktene er over forfiningsterskelen
        if sum(tre(i).Prediktor >= ovre*ones(2,1)) == 2
            forfine = [forfine; i];
        end
    end
    % Sjekk om randintervallene skal forfines
    if tre(i).Rand == 1
        if tre(i).Prediktor >= ovre == 1
            forfine = [forfine;i];
        end
    end
end
end

STATforfin = length(forfine);
for i = 1 : length(forfine)
    if tre(forfine(i)).Rand == 0
        tre = nyboks(tre,forfine(i),1);
        tre = nyboks(tre,forfine(i),2);
        tre(forfine(i)).Skjulsenter = 1;
    else
        tre = nyboks(tre,forfine(i),1);
    end
end
%
% ------- fjerner bokser
STATfjern = length(fjerne);
fjerne = sort(fjerne,'descend');
for i = 1 : length(fjerne)
       tre = fjernboks(tre,fjerne(i));
end

% STAT
data = hentdataA(tre);
STATs1 = length(data.sentre);

disp('------ STAT')
disp(['Antall sentre før: ' num2str(STATs0)])
disp(['Antall fjernes: ' num2str(STATfjern)])
disp(['Antall forfines: ' num2str(STATforfin)])
disp(['Antall sentre etter: ' num2str(STATs1)])

if STATforfin == 0
```

```
    stopp = 1;
end
teller = teller + 1;

end

data = hentdataA(tre);
sentre = data.sentre;

% hvis t=0 returneres verdier fra initialfunksjonen
if start == 1
    uN = initialu(sentrefor,0);
    sentre = sentrefor;
elseif start == 0
    uN = f(sentrefor);
    sentre = sentrefor;
end
```

Binartre.m

```
% Binartre.m - Matlab klasse.Implementering av binærtre.
%
% 1 x N datastruktur med N noder
classdef Binartre

properties
    Senter
    Dim
    Child
    Parent
    Nivaa
    Rand
    Sjekkpkt
    Sjekkpktkoord
    ID
    Epsilon = [];
    Prediktor = [];
    Nybokskoord = [0 0];
    Fjernboks = 0;
    Skjulsenter = 0;
    Indikator = [];
end

methods

function obj = initier(obj,N, epsilon)
```

```
a = -1;
b = 1;

bokslengde = (b - a)/(N-1);
senter = a  : bokslengde : b ;

obj(1).ID = 1;
obj(1).Senter = senter(1);
obj(1).Sjekkpkt = a + bokslengde/4;
obj(1).Sjekkpktkoord = 1;
obj(1).Rand = 1;
obj(1).Dim = bokslengde;
bj(1).Nivaa = 0;
obj(1).Epsilon = epsilon;

for i = 2 : length(senter)-1
    obj(i).ID = i;
    obj(i).Senter = senter(i);
    obj(i).Dim = bokslengde;
    obj(i).Child = [];
    obj(i).Parent = 0;
    obj(i).Nivaa = 0;
    obj(i).Rand = 0;
    obj(i).Epsilon = epsilon;
    obj(i).Sjekkpktkoord = [1,2];
end

obj(N).ID = N;
obj(N).Senter = senter(N);
obj(N).Sjekkpkt = b - bokslengde/4;
obj(N).Sjekkpktkoord = 1;
obj(N).Rand = 1;
obj(N).Dim = bokslengde;
obj(N).Nivaa = 0;
obj(N).Epsilon = epsilon;

for i = 2 : length(obj)-1
    obj(i).Sjekkpkt = finnsjekkpkt(obj(i));
end

end

function data = hentdataA(obj)
indresentre = [];
indreeps = [];
randsentre = [];
randeps = [];
indresjekkpkt = [];
randsjekkpkt = [];
```

```
iID = [];
rID = [];

for i = 1 : length(obj)
    if obj(i).Rand == 1
        randsentre = [randsentre; obj(i).Senter];
        randeps = [randeps; obj(i).Epsilon];
        randsjekkpkt = [randsjekkpkt; obj(i).Sjekkpkt];
        if size(obj(i).Sjekkpkt,1) > 0
            rID = [rID; obj(i).ID];
        end
    elseif obj(i).Rand == 0 && obj(i).Skjulsenter == 0
        indresentre = [indresentre; obj(i).Senter];
        indreeps = [indreeps; obj(i).Epsilon];
        indresjekkpkt = [indresjekkpkt; obj(i).Sjekkpkt];
        iID = [iID; ones(2,1)*obj(i).ID];
    end
end
data.indresentre = indresentre;
data.indreeps = indreeps;
data.randsentre = randsentre;
data.randeps = randeps;
data.indresjekkpkt = indresjekkpkt;
data.randsjekkpkt = randsjekkpkt;
data.iID = iID;
data.rID = rID;
data.sentre = [indresentre;randsentre];
data.eps = [indreeps;randeps];
data.sjekkpkt = [indresjekkpkt; randsjekkpkt];
data.ID = [iID;rID];
end

function obj = nyboks(obj,ID,plassering)
% function obj = nyboks(obj, ID, plassering)
% obj - Kvadtre objekt
% ID - Hver boks sin unike ID.
% plassering - hvilket sjekkpunkt som skal gjøres til senter i
% ny boks
a = plassering;
boks = obj(ID);
if boks.Rand == 1
    nyttsenter = boks.Sjekkpkt;
    nybokslengde = boks.Dim/2;

    nyboks = Binartre;
    nyboks.Senter = nyttsenter;
    nyboks.Dim = nybokslengde;
    nyboks.Child = [];
    nyboks.Nivaa = boks.Nivaa + 1;
```

```
        nyboks.Rand = 0;


        nyboks.Sjekkpkt(1,1) = nyboks.Senter(1,1) - nyboks.Dim/4;
        nyboks.Sjekkpkt(2,1) = nyboks.Senter(1,1) + nyboks.Dim/4;
        nyboks.Sjekkpktkoord = [1 2];
        nyboks.Parent = ID;

        nyboks.Epsilon = obj(ID).Epsilon*2; % nye boksens epsilon
        obj(ID).Epsilon = obj(ID).Epsilon*2; % gamle boksens epsilon

        obj = [obj, nyboks];
        obj(end).ID = length(obj);

        obj(ID).Child = length(obj);
        obj(ID).Sjekkpkt = [];
        obj(ID).Sjekkpktkoord = 0;

    else


        nyttsenter = boks.Sjekkpkt(boks.Sjekkpktkoord(a),1);
        nybokslengde = boks.Dim/2;

        nyboks = Binartre;
        nyboks.Senter = nyttsenter;
        nyboks.Dim = nybokslengde;
        nyboks.Child = [];
        nyboks.Nivaa = boks.Nivaa + 1;
        nyboks.Rand = 0;
        nyboks.Sjekkpkt = finnsjekkpkt(nyboks);
        nyboks.Sjekkpktkoord = [1 2];
        nyboks.Parent = ID;

        %epsilon. Sjekk om epsilon skal dobles. Parents epsilon
        %dobles kun ved første underboks
        str = size(obj(ID).Sjekkpkt,1);
        if str(1) == 2 % hvis parent ikke hadde underbokser
            nyboks.Epsilon = obj(ID).Epsilon*2;
            obj(ID).Epsilon = obj(ID).Epsilon*2;
        else
            nyboks.Epsilon = obj(ID).Epsilon;
        end

        % Fjerner sjekkpunktet i parentboksen
        c = obj(ID).Sjekkpktkoord(a);
        obj(ID).Sjekkpkt(c,:) = [];
        obj(ID).Sjekkpktkoord(a) = 0;
```

```
    for i = 1 : 2
        if obj(ID).Sjekkpktkoord(i) > c
            obj(ID).Sjekkpktkoord(i) = obj(ID).Sjekkpktkoord(i) - 1;
        end
    end

    obj = [obj, nyboks];
    obj(end).ID = length(obj);
    obj(ID).Child = [obj(ID).Child length(obj)];

    if length(obj(ID).Child) == 2;
        obj(ID).Skjulsenter = 1;
    end

end
end

function [obj, aktiver] = fjernboks(obj,ID)
% 1. Finner ID til parentboksen
% 2. Fjerner Child-referansen i parentboksen
% 3. "Aktiverer" senteret i parentboksen
% 4. Hvis parentboksen nå ikke har childs -> halvér epsilon
% 5. Legg til nytt sjekkpunkt i parentboksen
% 6. Oppdatér Sjekkpktkoord til parentboksen
if isempty(obj(ID).Child) == 0
    disp('Kan ikke fjerne boks!')
    disp(obj(ID))
else

aktiver = 0;

% function obj = fjernboks(obj, ID)
% fjener boks med gitt ID
parentID = obj(ID).Parent;
if parentID ~= 0
    b = find(obj(parentID).Child == ID);
    obj(parentID).Child(b) = []; % Fjerner Child-referansen i Parentboksen

    % Hvis parentboksen ikke har child (bortsett fra den som ble fjernet)
    if isempty(obj(parentID).Child) == 1
        obj(parentID).Epsilon = obj(parentID).Epsilon/2; %halveres epsilon
        obj(parentID).Skjulsenter = 0; % senteret aktiveres
        aktiver = 1;
    end
    % Legger til nytt sjekkpunkt i parent
    obj(parentID).Sjekkpkt = [obj(parentID).Sjekkpkt; obj(ID).Senter];

    if obj(parentID).Rand == 1
        if obj(parentID).Senter == -1
```

```
            temp = obj(parentID).Senter + obj(parentID).Dim/4;
        else
            temp = obj(parentID).Senter - obj(parentID).Dim/4;
        end
    else
        temp = finnsjekkpkt(obj(parentID));
    end
    b = find(temp(:,1) == obj(ID).Senter(1,1));
    obj(parentID).Sjekkpktkoord(b) = size(obj(parentID).Sjekkpkt,1);
end
% oppdatere boksenes ID slik at den stemmer overens med
% tabellreferansene
for i = ID+1 : length(obj)
    obj(i).ID = i-1; % Setter ID = ID - 1
end

% oppdatere parent og child verdier
for i = 1 : length(obj)
    if obj(i).Parent > ID
        obj(i).Parent = obj(i).Parent - 1;
    end
    for j = 1 : length(obj(i).Child)
        if obj(i).Child(j) > ID
            obj(i).Child(j) = obj(i).Child(j)-1;
        end
    end
end

obj(ID) = [];


end
end

function res = finnsjekkpkt(obj)
    a = obj.Senter - obj.Dim/4;
    b = obj.Senter + obj.Dim/4;
    res = [a;b];
end


end

end
```

lagindikator.m

```
function res = lagindikator(lambda, f, data, ddrbf)

L = ddrbf(lagepsilon(size(data.sjekkpkt,1),data.eps), ...
    distansematrise(data.sjekkpkt,data.sentre));
res = abs(L*lambda - f(data.sjekkpkt(:,1)));

end
```

distansematrise.m

```
% A = distansematrise(x,y)
%
% Setter sammen distansematrisen A_ij = ||x_i-y_j||_2
% inn: kolonnevektorene x og y
% ut: distansematrisen til x og y
function res = distansematrise(x, y)
    res = sqrt(differansematrise(x,y).^2);
end
```

differansematrise.m

```
% A = differansematrise(x,y)
%
% Setter sammen differansematrisen A_ij = x_i-y_j
% inn: kolonnevektorene x og y
% ut: A_ij = x_i-y_j
function res = differansematrise(x, y)
    [xx,yy] = ndgrid(x(:), y(:));
    res = xx-yy;
end
```

burgersledd.m

```
% burgersledd,
%
% brukes av Matlab ode15s
function dudt = burgersledd(t,u,Dx,Dxx,u1,b,u2,c)
    v = 0.0035;
    dudt = -u.*(Dx*u) + v*(Dxx*u);
    dudt(b) = u(b)-u1;
```

```
    dudt(c) = u(c)-u2;
end
```

initialu.m

```
% u = initialu(x,t)
%
% Eksakt løsning til Burgers ligning i kapittel 3. Brukes til å generere
% initialverdier og randverdier.
function res = initialu(x,t)
    v = 0.0035;
    a = @(x,t) -(x+0.5+4.95*t)/(20*v);
    b = @(x,t) -(x+0.5+0.75*t)./(4*v);
    c = @(x,t) -(x+0.625)./(2*v);
    u = @(x,t) (0.1*exp(a(x,t)) + 0.5*exp(b(x,t)) + exp(c(x,t)))./...
    (exp(a(x,t))+exp(b(x,t))+exp(c(x,t)));
    res = u(x,t);
end
```

lagepsilon.m

```
% res = lagepsilon(m,epsilon) -
% Brukes til å sette sammen RBF-matriser med individuell formparameter
%
% inn: m - tallverdi, epsilon - N vektor med formparametre
% ut: MxN matise med M kopier av vektoren epsilon
function res = lagepsilon(m,epsilon)
    for i = 1:length(epsilon)
        res(:,i) = ones(m,1)*epsilon(i);
    end
end
```

Bibliografi

[1] V. Bayona, M. Moscoso, and M. Kindelan. Optimal variable shape parameter for multiquadric based RBF-FD method. *Journal of Computational Physics*, 231(6):2466–2481, 2012.

[2] J. Behrens and A. Iske. Grid-free adaptive semi-Lagrangian advection using radial basis functions. *Computers & Mathematics with Applications*, 2002.

[3] W. E Boyce. *Elementary Differential Equations and Boundary Value Problems*. Wiley, 2008.

[4] J. P. Boyd. *Chebyshev and Fourier spectral methods*. DOVER Publications, 2001.

[5] J. P. Boyd and K. W. Gildersleeve. Numerical experiments on the condition number of the interpolation matrices for radial basis functions. *Applied Numerical Mathematics*, 61(4):443–459, 2011.

[6] T. Driscoll and A. R. H. Heryudono. Adaptive residual subsampling methods for radial basis function interpolation and collocation problems. *Computers & Mathematics with Applications*, 53(6):927–939, 2007.

[7] G. E. Fasshauer. *Meshfree Approximation Methods with MATLAB*. World Scientific, 2008.

[8] B. Fornberg, T. Driscoll, G. Wright, and R. Charles. Observations on the behavior of radial basis function approximations near boundaries. *Computers & Mathematics with Applications*, 43(3-5):473–490, 2002.

[9] B. Fornberg and G. Wright. Stable computation of multiquadric interpolants for all values of the shape parameter. *Computers & Mathematics with Applications*, 48(5-6):853–867, 2004.

[10] Y. C. Hon, R. Schaback, and X. Zhou. An adaptive greedy algorithm for solving large RBF collocation problems. *Numerical Algorithms*, pages 13–25, 2003.

[11] E. J. Kansa. Multiquadratics - A Scattered Data Approximation Scheme With Applications To Computation Fluid-Dynamics-I. 19(8):127–145, 1990.

[12] E. J. Kansa. Multiquadratics - A scattered data approximation scheme with applications to computational fluid-dynamics II. *Computers & Mathematics with Applications*, 19(8):147–161, 1990.

[13] D Kincaid and W Cheney. *Numerical Analysis: Mathematics of Scientific Computing*. BROOKS/COLE, third edition, 2002.

[14] A. Munoz, P. Gonzalez, and G. Rodriguez. Adaptive node refinement collocation method for partial differential equations. *Seventh Mexican International Conference on Computer Science*, pages 70–77, 2006.

[15] S. Sarra. Adaptive radial basis function methods for time dependent partial differential equations. *Applied Numerical Mathematics*, 54(1):79–94, 2005.

[16] D. Shepard. A two-dimensional interpolation function for irregularly-spaced data. *Proceedings of the 1968 23rd ACM national conference*, pages 517–524, 1968.

[17] L. N. Trefethen. *Spectral Methods in MATLAB*. Society for Industrial and Applied Mathematics, 2000.

[18] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, 1997.