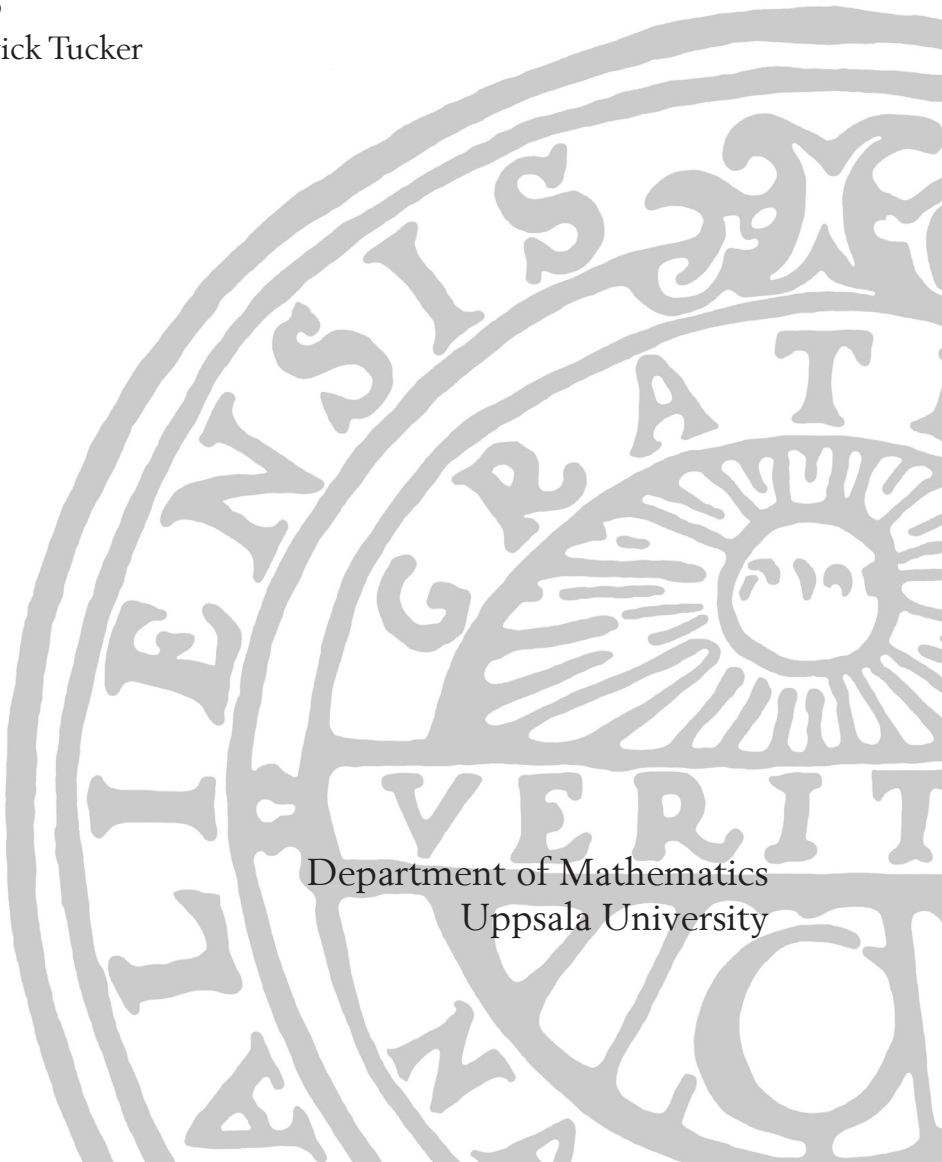
UPPSALA
UNIVERSITET

U.U.D.M. Project Report 2014:41

Datorstödda bevis

Robin Samuelsson

Examensarbete i matematik, 15 hp
Handledare och examinator: Warwick Tucker
Juli 2014



Department of Mathematics
Uppsala University

Sammanfattning

Att arbeta med datorer för matematiska bevis kräver ofta en stor noggranhet och precision vilket, på grund av datorers begränsningar i att representera data, ofta är svårt att uppnå med reella analysmetoder.

Det här arbetet beskriver hur den precisionen istället kan uppnås genom att man arbetar med intervallaritmetik och intervallanalys i datorn. Stora delar av arbetet handlar därför om vilka överlagringar som krävs i Matlab för att bygga ett datorsystem som kan arbeta i intervallanalys geom att presentera de koder som krävs för detta.

Arbetet avslutas med en beskrivning av en kraftig intervallbaserad metod vilken kan användas för att bevisa existenser eller icke-existenser av nollställen till funktioner.

Innehåll

1 Bakgrund

Allt sedan de första datorerna började utvecklas har antalet transistorer hos datorns processor ökat med en oerhörd takt vilket direkt påverkar beräkningshastigheten hos datorn. Dess ökning kan till och med beskrivas vara exponentiell med en fördubbling av antalet transistorer vartannat år (något som numera är känt som Moores lag).

Dagens Industri publicerade under det gågna året, 2013, en artikel där en undersökning av superdatorers prestanda hade undersökts. En av de undersökta datorerna, den kinesiska superdatorn Tianhe-2 visade sig kunna utföra otroliga 1000 biljoner beräkningar per sekund.[1] Men hur väl utförs dessa många beräkningar? Frågan som Warwick Tucker ställer sig i inledningen av sin bok Validated numerics beskriver problematiken med att fokusera allt för mycket på prestandan utan att egentligen förbättra precisionen av beräkningarna: '[a]re we just getting the wrong answers faster?'[2]. Det är här intervallaritmetikens fördelar kommer in i arbetet med datorstödda bevis.

Det som utgör grunden till mitt arbete är intervallaritmetiken samt intervallanalysen och jag kommer således ägna en del tid åt att ge en bakgrund samt beskriva denna. Det finns inte så många källor som beskriver validerad numerik och intervallanalys på en bredare nivå men samtidigt mer grundläggande nivå och jag har därför i mitt arbete uteslutande använt mig av följande två verk: *Validated Numerics* (2011) av Warwick Tucker samt *Introduction to Interval Analysis* (2009) av Ramon E. Moore, R. Baker Kearfott och Michael J. Cloud vilka kommer att refereras till som [Tu11] respektive [Mo09]. Beteckningar och begrepp ligger emellertid närmare Tuckers arbete än Moores, till exempel så har jag valt att använda tjocka bokstäver för att beteckna intervall istället för, likt Moore, använda versaler.

Beräkningar med intervall har förekommit långt tillbaka i tiden. Man vet att redan under 200 f.v.t. använde Arkimedes en övre och en undre gräns i ett intervall för att beskriva π efter att ha studerat en 96-gon.[3] I vår nutid har intervallanalysen fått en allt mer framträdande roll i datorberäkningar eftersom den, till skillnad från flyttalsaritmetik, tillför den precision som krävs vid beräkningar som vid små initiala fel kan ge oerhörda konsekvenser vid slutresultatet som till exempel beräkningar av kaotiska dynamiska system. Ett exempel där intervallaritmetik hade varit behjälplig, hämtat ur R. E. Moores bok Introduction to Interval Analysis, ges nedan:

[1] http://www.di.se/artiklar/2013/6/17/kina-har-snabbaste-superdatorn/, kontrollerad 18-06-2014

[2] [Tu11],s. ix

[3] [Mo09], s. 1

Givet rekursionsformeln

$$x_{n+1} = (x_n)^2, \quad (n = 0, 1, 2, ...)$$

, och antag att

$$x_0 = 1 - 10^{-21}$$

Om vi söker x_{75} genom 10-platsaritmetik med dator så kommer vi att erhålla de approximativa värdena $x_0 = 1, x_1 = 1, ..., x_{75} = 1$ eftersom värdena i beräkningen förhåller sig så pass nära 1 att de approximeras till detta.
Söker vi samma x men med 20 platser istället så får vi samma resultat. Detta trots att det exakta värdet uppfyller $x_{75} < 10^{-10}$. För att få fram ett exakt resultat krävs fler platser eller, något som kommer att utnyttjas i det här arbetet: att man använder sig utav intervall för att bestämma svaret mer exakt.

Mycket av arbetet med den moderna intervallanalysen kan sägas ha börjat i och med Ramon E. Moores publiceringar på ämnet under 60-talet.[4] Moores forskning kom under en tid då datorn fortfarande befann sig i ett primitivt stadium men likväl publicerade han 1959 en rapport om hur en eventuell användning av intervallaritmetik skulle kunna implementeras på en dator, något som resulterade i ett program som kunde begränsa lösningar till ordinära differentialekvationer. Moores arbete inspirerade sedermera andra forskare som bland annat tog fram en aritmetik och analys för komplexa intervall.[5]

Arbetet med att standardisera beräkningar med intervall pågår i skrivande stund genom IEEE (arbetet kan följas på http://grouper.ieee.org/groups/1788/).[6]

[4][Mo09], s. 16-17

[5]$http://interval.louisiana.edu/Moores_early_papers/bibliography.html, senastkontrollerad 18-06-2014$

[6][Tu11] s. 37

2 Datorn

Datorer består av flertalet komponenter men den del som kan vara intressant för det här arbetet är processorn där själva beräkningarna utförs via flyttal. Processorn kan i sig delas upp i ett antal komponenter där ALU:n (aritmetikenheten) utför de enklare beräkningarna vilka sedan skickas vidare som data via styrenheten. Styrenheten styr då flödet av data mellan processorn och andra apparaturer. Då datorn räknar i så kallade flyttal bearbetas emellertid beräkningarna i flyttalsprocessorn. Mängden data som processorn kan flytta i varje arbetscykel beror på vilken typ av processorarkitektur datorn använder, till exempel ett 64- eller ett 32-bitars system.

Eftersom datorn inte kan representera oändligt många tal används approximeringar av de reella talen, det är dessa representationer av de reella talen som benämns som flyttal och sker oftast i en vald bas, exempelvis i basen 2 vars system benämns som binärt.

Flyttalen består av fyra komponenter: tecken, mantissa, bas och exponent. Tecknet representerar antingen + eller - och följs av mantissan som representerar själva siffrorna i positionerna. Mantissan kan dessutom vara normaliserad vilket innebär att heltalssidan om decimaltecknet endast består av en siffra (exempel: $6542.1 \implies 6.5421$ eller $101101.1 \implies 1.011011$) vilken i det binära talsystemet ignoreras då den ändå alltid är 1 vilket sparar minne för datorn. Mantissan följs sedan av basen och exponenten som avgör vilket typ av system det är samt hur många steg decimaltecknet har flyttats i mantissan, till exempel ger $101101.1 \implies 1.011011$ exponenten 5. Om exponenten dessutom är bias så adderas 127 till exponent-talet för att på så sätt undkomma problematiken med negativa exponenter så att till exempel exponenten -9 ger det positiva talet 118. [7]

2.1 Avrundningar

Datorer använder oftast IEEE:s standard, det vill säga de arbetar med flyttal i basen två. Då ett reellt tal i bas 10 matas in i datorn så kommer detta översättas till närmsta flyttal i den gällande basen för datorsystemet. För att intervallaritmetiken och intervallanalysen ska garantera inklusion av de korrekta områdena, exempelvis värdemängden till en viss funktion för en viss definitionsmängd, så krävs det att datorn avrundar inkluderande istället för exkluderande. Detta kan göras med riktad avrundning. Det finns ett antal typer av riktade avrundningar, fyra av dessa kommer att användas för att bygga ett datorsystem som arbetar med intervallanalys (se avsnittet Programko-

[7] [Tu11], s. 1-5

der). Dessa fyra är avrundning mot oändligheten, mot minus oändligheten, mot noll och till närmsta flyttal:

Då den utvidgade mängden av flyttal definieras som $\mathbb{F}* = \mathbb{F} \cup \{-\infty, +\infty\}$, det vill säga där $\mathbb{F}*$ är mängden för flyttal tillsammans med de negativa och positiva oändligheterna, så gäller

- Avrunda mot $-\infty$: avrundning sker mot närmsta flyttal som ligger mot $-\infty$ vilket kan definieras på följande sätt: $\nabla(x) = max\{y \in \mathbb{F}* : y \leq x\}$

- Avrunda mot $+\infty$: avrundning sker mot närmsta flyttal som ligger mot $+\infty$ vilket kan definieras på följande sätt: $\triangle\ (x) = max\{y \in \mathbb{F}* : y \geq x\}$

- Avrunda mot 0: trunkering som definieras genom $\square(x) = sign(x)max\{y \in \mathbb{F}* : y \leq |x|\}$

- Avrunda till närmsta flyttal: även om det sökta svaret inkluderas i slutintervallet så kan felet i avrundningarna bli lika stort som avrundningen av talet x mot minus oändligheten till avrundningen av talet x mot plus oändligheten men om man istället avrundar till närmsta flyttal så kan man undvika allt för stora avrundningsfel. Hur avrundning mot närmsta flyttal ska ske beror på N^n_{max} (det största normaliserade flyttalet) vilket ger ett φ som ger hur avrundningen ska genomföras:

Först avgörs φ:

Om $|x| \leq N^n_{max} \rightarrow \varphi = 0.5(\triangle\ (x) + \nabla(x))$

Om $|x| \geq N^n_{max} \rightarrow \varphi = sign(x)N^n_{max}$

Utifrån vad φ är kan sedan avrundning göras:

(1) Om $x < \varphi$ så sker avrundning genom $\nabla(x)$

(2) Om $x > \varphi$ så sker avrundning genom $\triangle\ (x)$

(3) Om $x = \varphi$ så kan avrundning ske utifrån två olika alternativ. Det senare kommer dock bara att definieras här eftersom det är sättet som man avrundar på i de flesta datorer som används för vardagligt bruk. [8]

[8] [Tu11], s. 8

(3.1) Låt mantissorna av $\nabla(x)$ och $\triangle$ vara $(a_0.a_1a_2a_3...a_{p-1})_\beta$ och $(b_0.b_1b_2b_3...b_{p-1})_\beta$. Så länge x inte är ett tal i den utvidgade mängden av flyttal så måste, enligt lemma 1.10 i [Tu11], precis ett av de två avslutande siffrorna eller enheterna i mantissorna för $\nabla(x)$ och $\triangle$ vara jämn. På så sätt kan alltid följande fall uppfyllas:

$$x > 0 \rightarrow \square(x) = \begin{cases} \nabla(x), & \text{om} \quad x \in [\nabla(x), \varphi), \quad \text{eller om} \quad x = \varphi \quad \text{och} \quad a_{p-1} \quad \text{är jämn,} \\ \triangle\,(x), & \text{om} \quad x \in (\varphi, \triangle\,(x)], \quad \text{eller om} \quad x = \varphi \quad \text{och} \quad b_{p-1} \quad \text{är jämn,} \end{cases}$$

$x < 0 \rightarrow \square(x) = -\square(-x)$.[9]
Avrundningarna används i de överlagringar i Matlab som beskrivs under sektionen Programkoder (börjar i setround-filen).

2.2 Intervallaritmetik för flyttal

Intervallaritmetiken kommer att beskrivas under nästa rubrik och beskrivs där för $\mathbb{R}$. Nedan följer alltså en beskrivning av intervallaritmetiken för flyttal.

För flyttal arbetar man över flyttalsmängden $\mathbb{F}$ vilket ger en annan uppsättning regler för de aritmetiska operationerna eftersom $\mathbb{F}$ och $\mathbb{IF}$ (mängden av alla intervall med ändpunkter i $\mathbb{F}$) båda är ändliga mängder. Den tidigare är dock inte aritmetiskt sluten vilket medför att den nedre gränsen i intervalloperanden måste avrundas nedåt och den övre gränsen måste avrundas uppåt till närmaste flyttal för att det sökta resultatet garanterat ska ligga i det resulterande intervallet.

Aritmetiska operationer mellan två intervall $\mathbf{a}$ och $\mathbf{b}$ tagna från $\mathbb{IF}$ utförs på följande sätt:

$$\mathbf{a} + \mathbf{b} = [\nabla(\underline{a} + \underline{b}), \triangle\,(\overline{a} + \overline{b})]$$
$$\mathbf{a} - \mathbf{b} = [\nabla(\underline{a} + \overline{b}), \triangle\,(\overline{a} + \underline{b})]$$
$$\mathbf{a} \times \mathbf{b} = [\min\{\nabla\underline{a}\underline{b}, \nabla\underline{a}\overline{b}, \nabla\overline{a}\underline{b}, \nabla\overline{a}\overline{b}\}, \max\{\triangle\,\underline{a}\underline{b}, \triangle\,\underline{a}\overline{b}, \triangle\,\overline{a}\underline{b}, \triangle\,\overline{a}\overline{b}\}]$$
$$\mathbf{a} \div \mathbf{b} = [\min\{\nabla(\underline{a}/\underline{b}), \nabla(\underline{a}/\overline{b}), \nabla(\overline{a}/\underline{b}), \nabla(\overline{a}/\overline{b})\}, \max\{\triangle\,(\underline{a}/\underline{b}), \triangle\,(\underline{a}/\overline{b}), \triangle\,(\overline{a}/\underline{b}), \triangle\,(\overline{a}/\overline{b})\}],$$
$$\text{om} \quad 0 \notin b.$$

[10]

[9] [Tu11], s. 8-9
[10] [Tu11], s. 37

3 Grundläggande teorier vid beräkningar med intervall

Att använda sig av intervall istället för tal är särskilt effektivt när det kommer till att erhålla precisa slutresultat. Att en längd exempelvis kan uppskattas till 2 meter fastän den exakta längden är 1.95 meter ger att påståendet 'längden är 2 meter' är falskt medan påståendet 'längden är mellan 1.945 och 1.955 meter' är sant vilket medför att man kan uppskatta felet i beräkningen istället för att anta det approximerade värdet som kan ge stora fel vid större beräkningar.

För att kunna räkna med intervall krävs en uppsättning regler för mängdteori med intervall samt intervallaritmetik:

3.1 Mängdteori för intervall

Låt $\mathbf{a}$ och $\mathbf{b}$ i fortsättningen vara intervall, i det här fallet på sådant sätt att

$$\mathbf{a} = [\underline{a}, \overline{a}] = \{x \in \mathbb{R} : \underline{a} \leq x \leq \overline{a}\},$$
$$\mathbf{b} = [\underline{b}, \overline{b}] = \{x \in \mathbb{R} : \underline{b} \leq x \leq \overline{b}\}.$$

Då gäller följande regler

$$\mathbf{a} = \mathbf{b} \iff \underline{a} = \underline{b} \quad \text{och} \quad \overline{a} = \overline{b}$$
$$\mathbf{a} \subseteq \mathbf{b} \iff \underline{b} \leq \underline{a} \quad \text{och} \quad \overline{a} \leq \overline{b}$$
$$\mathbf{a} \subset \mathbf{b} \iff \mathbf{a} \subseteq \mathbf{b} \quad \text{och} \quad \mathbf{a} \neq \mathbf{b}.$$

Att $\mathbf{a} \leq \mathbf{b}$ innebär till detta att de båda gränserna i $\mathbf{a}$ är mindre än eller lika med respektive gräns i $\mathbf{b}$. Utöver detta gäller för intervall även följande för att inkludera de fall då unionen av två intervall bildar ett nytt intervall (det vill säga även vid de fall då $\mathbf{a} \cap \mathbf{b} = \emptyset$)

$$\mathbf{a} \sqcup \mathbf{b} = [\min\{\underline{a}, \underline{b}\}, \max\{\overline{a}, \overline{b}\}]$$

vilket alltså är ett nytt intervall.[11] [12]

3.2 Intervallaritmetiken

När det kommer till att räkna med intervall gäller konventionen att enskilda reella tal identifieras som så kallade degenererade intervall, alltså ett intervall

[11] [Mo09], s. 10
[12] [Tu11], s. 25-26

där $\underline{a} = \overline{a}$ så att $a = \mathbf{a} = [\underline{a}, \overline{a}]$

Definition 1
Om $\mathbb{IR}$ är definierat som mängden av alla reella intervall, $\mathbf{a}$ och $\mathbf{b}$ två intervall och $\odot$ representerar en av operatorerna $+, -, \times, \div$, så är aritmetik över elementen i $\mathbb{IR}$ definierat som

$$\mathbf{a} \odot \mathbf{b} = a \odot b : a \in \mathbf{a}, b \in \mathbf{b},$$

$$\mathbf{a} \div \mathbf{b} \quad \text{är odefinierat om} \quad 0 \in \mathbf{b}.$$[13]

Utifrån definitionen ges propositionen (*Proposition 1*)[14]

$$\mathbf{a} + \mathbf{b} = [\underline{a} + \underline{b}, \overline{a} + \overline{a}]$$
$$\mathbf{a} - \mathbf{b} = [\underline{a} - \overline{b}, \overline{a} - \underline{b}]$$
$$\mathbf{a} \times \mathbf{b} = [\min\{\underline{ab}, \underline{a}\overline{b}, \overline{a}\underline{b}, \overline{ab}\}, \max\{\underline{ab}, \underline{a}\overline{b}, \overline{a}\underline{b}, \overline{ab}\}]$$
$$\mathbf{a} \div \mathbf{b} = \mathbf{a} \times [1/\overline{b}, 1/\underline{b}], \qquad \text{om} \quad 0 \notin \mathbf{b}$$

vilket alltså utgör grunden för beräkningar med intervall.

Man ser här att addition och multiplikation i intervallaritmetik är både associativa och kommutativa men saknar invers eftersom division och subtraktion inte är direkta inversa operationer till addition och multiplikation. Exempelvis gäller ej $[2, 5] \div [2, 5] = [1, 1]$ eftersom detta, enligt proposition 1, blir

$$[2, 5] \div [2, 5] = [2, 5] \times [1/5, 1/2] =$$
$$[\min\{2/5, 2/2, 5/5, 5/2\}, \max\{2/5, 2/2, 5/5, 5/2\}] = [2/5, 5/2]$$

Detta ger flera konsekvenser som beskrivs senare, bland annat att den distributiva lagen ej gäller för intervallaritmetik och intervallanalys.

3.3 Algebraiska egenskaper hos intervallaritmetiken

Vid beräkningar med intervall gäller ej alltid distributiva lagen på grund av att division och subtraktion saknar reciprokalerna så att $(1/\mathbf{x}) \times \mathbf{x} \neq 1$ för alla $\mathbf{x} \in \mathbb{IR}$ och $\mathbf{x} - \mathbf{x} \neq 0$ för alla $\mathbf{x} \in \mathbb{IR}$ (undantag finns då $\mathbf{x}$ är ett

[13] [Tu11], s. 27
[14] [Tu11], s. 27 (med bevis)

degenererat intervall). Det här betyder alltså att om t, u och v är godtyckliga tal så gäller vanligtvis $t(u+v) = tu + tv$ vilket ej gäller i intervallaritmetik. För intervallaritmetik gäller istället det som benämns sub-distribution:
Givet det godtyckliga intervallet $\mathbf{c}$ gäller följande;

$$\mathbf{a}(\mathbf{b}+\mathbf{c}) \subseteq \mathbf{ab} + \mathbf{ac}$$

Ett intervall definieras som symmetriskt då $\underline{a} = -\overline{a}$ vilket även ger att

$$\underline{a} = -\overline{a} \iff mid[\underline{a}, \overline{a}] = 0.$$

Generellt gäller här att $mid[\underline{a}, \overline{a}] = (\underline{a} + \overline{a})/2$. som vid symmetri ger att $mid[\underline{a}, \overline{a}] = (\underline{a} + \overline{a})/2 = (\underline{a} + (-\underline{a}))/2 = 0/2 = 0$.[15]
Den enda kancelleringslag som håller i intervallaritmetik är den då för intervallen $\mathbf{a}, \mathbf{b}$ och $\mathbf{c}$ vid addition på så sätt att

$$\mathbf{a} + \mathbf{c} = \mathbf{b} + \mathbf{c} \implies \mathbf{a} = \mathbf{b}$$

Vid beräkningar med degenererade intervall gäller emellertid fortfarande även den multiplikativa kancelleringen.

Sats 1
Slutligen har vi satsen som säger att intervallaritmetiken är inklusionsisotonisk, det vill säga att om $\odot$ låtes vara de fyra operationerna $+, -, \div, \times$ och $\mathbf{a}, \mathbf{b}, \mathbf{c}$ och $\mathbf{d}$ är intervall på sådant sätt att

$$\mathbf{a} \subseteq \mathbf{c} \wedge \mathbf{b} \subseteq \mathbf{d}$$

så är inklusionsisotonicitet

$$\mathbf{a} \odot \mathbf{b} \subseteq \mathbf{c} \odot \mathbf{d}.$$[16]

3.4 Övriga viktiga uttryck för intervallberäkning

Andra uttryck som kan vara viktiga att hålla reda på är

$$rad(\mathbf{x}) = \frac{1}{2} \times (\overline{x} - \underline{x}) \qquad \text{(radien av} \quad \mathbf{x})$$
$$mid(\mathbf{x}) = \frac{1}{2} \times (\overline{x} + \underline{x}) \qquad \text{(mittpunkten av} \quad \mathbf{x})$$
$$mig(\mathbf{x}) = \min\{|x| : x \in \mathbf{x}\} \qquad \text{(mignituden av} \quad \mathbf{x})$$
$$mag(\mathbf{x}) = \max\{|x| : x \in \mathbf{x}\} \qquad \text{(magnituden av} \quad \mathbf{x})$$

[15] [Mo09], s. 33-34
[16] Fullständigt bevis finns i [Tu11], s. 30 samt [Mo09], s. 34-35

4 Grundläggande programkoder

För att kunna använda intervallaritmetiken i exempelvis MATLAB (vilket jag har valt att göra) så måste man överlagra de redan inbyggda aritmetiska funktionerna i MATLAB.
Överlagring sker genom att man skapar .m-filer (funktioner) som sparas som de inbyggda aritmetiska funktionerna, alltså plus.m eller minus.m. Till detta behövs dessutom ett gäng andra filer för att exempelvis kunna skapa intervall, arbeta med degenererade intervall och runda av intervallgränser.

Det är emellertid viktigt att notera hur känsligt MATLAB är gällande var och hur mapparna som innehåller .m-filerna ligger. En bra och fungerande ordning är till exempel MATLAB > @interval > private där @interval innehåller filerna plus.m, mtimes.m, mrdivide.m, minus.m, interval.m och display.m. I mappen private läggs sedan cast.m, roundup.m, rounddown.m och setround.m (om setround.m fungerar så behövs ej roundup.m eller rounddown.m och vice versa).

4.1 Aritmetik

Nedan följer de grundläggande programkoder som krävs för att använda intervallaritmetik i MATLAB. Kommentarer är inbäddade i koderna efter %-tecken och ger en mer utförlig beskrivning om varje kod. De mest grundläggande koderna (utan kommentarer) är skrivna av Warwick Tucker och har då en hänvisning genom fotnot till ursprungskällan. Detta eftersom basen i överlagringarna är svår att göra på fler sätt, det man möjligen kan ändra är beteckningar för parametrar i koderna.
Interval.m [17]

```
1  function iv = interval(lo, hi)
2  %  Naiv funktion för att skapa intervall. Naiv på det ...
       sättet att funktionen ej
3  %  fungerar när hi < lo gäller (den typ av problem som ...
       affin utvidgning kan
4  %  hantera.
5  if nargin == 1
6      hi = lo;
7  elseif ( hi < lo )
8      error('Ändpunkterna definierar inte ett intervall. ')
9  end
10 iv.lo = lo; iv.hi = hi;
```

[17] [Tu11], s. 38

```
11 iv = class(iv,'interval');
12
13 end
```

Cast.m [18]

```
1  function [ a, b] = cast( a, b )
2  % Gör om alla icke-intervall till intervall.
3  if ~isa(a, 'interval')
4      a = interval(a);
5  end
6  if ~isa(b, 'interval')
7      b = interval(b);
8  end
9
10
11 end
```

Display.m [19]

```
1  function display (iv)
2  % En enkel output-formaterare för intervallklassen
3  disp([inputname(1), ' = ']);
4  fprintf(' [%17.17f, %17.17f]\n', iv.lo, iv.hi);
5
6  end
```

Setround.m [20]

```
1  function setround(rnd)
2  % En switch för att ändra avrundningsläge. Argumenten ...
       {+inf, -inf, 0.5, 0}
3  % korresponderar till avrundningarna {uppåt, nedåt, till ...
       närmsta flyttal, mot
4  % noll}.
5  % Problem med funktionen kan förekomma på olika ...
       plattformar. Det här kan lösas
```

[18] [Tu11], s. 40
[19] [Tu11], s. 39
[20] [Tu11], s. 40

```
% genom att istället använda sig av, de inte lika ...
    effektiva funktionerna,
% roundup.m och rounddown.m.
system_dependent('setround', rnd);


end
```

Roundup.m:

```
function y = roundup(x)
% Bristfällig funktion för att avrunda till närmaste ...
    flyttal uppåt. Ersätter då
% Setround.m.
if x >= 0
    y = x*(1+10^-15);
else
    y = x*(1-10^-15);

end
end
```

Rounddown.m:

```
function y = rounddown( x )
% Bristfällig funktion för att avrunda till närmaste ...
    flyttal nedåt. Ersätter då
% Setround.m
if x >= 0
    y = x*(1-10^-15);
else
    y=x*(1+10^-15);

end
end
```

Sedan följer själva överlagrings-funktionerna. Det bör ännu en gång påpekas hur viktigt det är med position av mappar för funktionerna. Exempelvis fungerar inte setround-funktionen om den inte ligger i undermappen private (något som jag missade och därmed fick mig att använda alternativen med roundup och rounddown i början). Här har jag i alla fall valt att använda

setround.m eftersom det fungerade men om ens system ej är kompatibelt med setround-funktionen så kan man undkomma detta genom att använda roundup.m och rounddown.m (alternativt skapa och kompilera en fil från C). Om man vill använda dessa istället så lägger man in avrundningsfunktionen efter varje beräkning (nedåt efter den första beräkningen och uppåt efter den andra) till skillnad från setround.

Nedan följer de koderna för de aritmetiska operationerna. Jag ger ett exempel på användandet av roundup och rounddown för plus.m bara för att visa på skillnaderna i var funktionen läggs in, därav förekommer plus.m två gånger:

Plus.m [21] (setround används):

```
1  function result = plus( a, b )
2  %Överlagra x-operatorn för intervall. Avrundning sker med ...
       setround-funktionen.
3  [a, b] = cast(a, b);
4  setround(-inf);
5  lo = a.lo + b.lo;
6  setround(+inf);
7  hi = a.hi + b.hi;
8  setround(0.5);
9  result = interval(lo, hi);
10
11 end
```

Plus.m (roundup och rounddown används):

```
1  function result = plus( a, b )
2  %Överlagra x-operatorn för intervall. Avrundning sker med ...
       roundup och rounddown.
3  [a, b] = cast(a, b);
4  lo = a.lo + b.lo;
5  rounddown(lo);
6  hi = a.hi + b.hi;
7  roundup(hi)
8  result = interval(lo, hi);
9
10 end
```

Minus.m:

[21] [Tu11], s. 39

```
function result = minus(a, b)
%Överlagra minus-funktionen så att den blir ...
    intervallaritmetisk
[a, b] = cast(a, b);
setround(-inf);
lo = a.lo - b.hi;
setround(+inf);
hi = a.hi - b.lo;
setround(0.5);
result = interval(lo, hi);

end
```

Mtimes.m:

```
function result = mtimes(a, b)
%Multiplikation med intervall
[a, b] = cast(a, b);
intve = [a.lo*b.lo a.lo*b.hi a.hi*b.lo a.hi*b.hi];
    setround(-inf);
    lo = min(intve);
    setround(+inf);
    hi = max(intve);
  setround(0.5);
    result = interval(lo, hi);

end
```

Mrdivide.m [22]

```
function result = mrdivide(a, b)
% Icke-optimal algoritm med division som ej fungerar med 0 ...
    i ett intervall.
[a, b] = cast(a, b);
if ( (b.lo <= 0.0) && (0.0 <= b.hi) )
    error('Denominator straddles zero.');
else
    intve = [a.lo/b.lo a.lo/b.hi a.hi/b.lo a.hi/b.hi];
    setround(-inf);
    lo = min(intve);
    setround(+inf);
    hi = max(intve);
```

[22] [Tu11], s. 40

```
12      setround(0.5);
13      result = interval(lo, hi);
14  end
15  end
```

Nu kan man även testa sub-distributionen hos intervallaritmetik.
Låt exempelvis a = interval(-2, -1), b = interval(-3, -2) och c = interval(1, 2).
Då får vi följande i MATLAB för a*(b+c) och a*b + a*c:
>> a*(b+c), a*b + a*c ans = [-0.00000000000000000, 4.00000000000000000] ans = [-2.00000000000000000, 5.00000000000000000]. Här ser vi alltså att vi fått ett konkret exempel för sub-distribution; $a * (b + c) \subseteq a * b + a * c$.

4.2 Relationer

För att få ett komplett grundläggande system för beräkningar med intervall behöver vi även överlagra de relationella operationerna.

Vi börjar med följande fyra booleska funktioner (funktioner som ger resultaten '1' eller '0' vilket representerar sant eller falskt):

Eq.m [23](likhet):

```
1  function result = eq(a,b)
2  % Överlagring av likthetsfunktionen '=='.
3  [a, b] = cast(a, b);
4  result = ( (a.lo == b.lo) & (a.hi == b.hi) );
5
6  end
```

Ne.m [24](ej likhet):

```
1  function result = ne( a,b )
2  % Överlagring för 'ej lika med'-funktionen '~='.
3  [a, b] = cast(a, b);
4  result = ( (a.lo ~= b.lo) | (a.hi ~= b.hi) );
5
6  end
```

Le.m [25](delmängd av):

[23][Tu11], s. 42
[24][Tu11], s. 42
[25][Tu11], s. 43

```
function result = le(a, b)
% Delmängdsfunktionen ('<=') överlagring.
[a, b] = cast(a, b);
result = ( (b.lo <= a.lo) & ( a.hi <= b.hi) );

end
```

Lt.m [26](Strikt delmängd av):

```
function result = lt( a, b )
% Överlagrade funktionen för 'strikt delmängd av', '<'.
[a, b] = cast(a, b);
result = ( ( b.lo < a.lo) & (a.hi < b.hi) );

end
```

Funktionen för att uttrycka union blir lite problematisk för intervall eftersom två intervall som ska sammanfogas till ett nytt intervall kan resultera i ett intervall med ett tomt gap i om inte intervallen skär varandra. Det här löses genom att man använder en 'hull' ($\sqcup$) istället för $\cup$. Denna fungerar på följande sätt:
givet intervallen $a = [2, 3]$ och $b = [5, 6]$ (som alltså ej skär varandra) så har man att $a \sqcup b = [2, 6]$. Generellt: $\mathbf{a} \sqcup \mathbf{b} = [\min\{\underline{a}, \underline{b}\}, \max\{\overline{a}, \overline{b}\}]$. För att sedan se om en intersektion är tom i Matlab så kan man använda dess inbyggda funktion 'isempty(X)' vilket ger antingen '1'/sant eller '0'/falskt.

Nedan följer då funktionskoderna för överlagring av eller- och och-funktionerna som här representerar intervallrelationerna $\sqcup$ och $\cap$. And.m [27](intersektion):

```
function result = and( a, b )
% Överlagring av och-funktionen '&' som i det här fallet är
% intersektionsrelationen.
[a, b] = cast(a, b);
if ( (a.hi < b.lo) | (b.hi < a.lo) )
    warning('The intervals do not intersect.');
    result = [];
else
    result = interval(max(a.lo, b.lo), min(a.hi, b.hi));
end
end
```

[26][Tu11], s. 43
[27][Tu11], s. 43

Or.m [28]('hull'):

```
1 function result = or( a, b )
2 % Överlagring av eller-funktionen '|' så att den blir ...
      'hull'-funktionen.
3 [a, b] = cast(a, b);
4 result = interval(min(a.lo, b.lo), max(a.hi, b.hi));
5
6 end
```

[28][Tu11], s. 44

5 Utvidgad intervallaritmetik

Ett problem i intervallaritmetiken som kan vara bra att kunna kringgå är division med noll. Detta eftersom man många gånger kan behöva dividera med ett intervall som innehåller noll vilket i enlighet med definition 1 ej är möjligt.

Problematiken häri består av att man då ej kan ha ett intervall med element både från den negativa och positiva delen av den reella talmängden som nämnare eftersom detta även kommer att inkludera elementet 0 i mängden. Man kan dock kringgå detta genom att utvidga intervallaritmetiken till att även inkludera oändlighet (konkret genom en utvidgning av $\mathbb{R}$).

Det finns två sätt att göra detta på, varav den ena dessutom har en tillagd egenskap för att kunna tilldela dess element unika reciprokaler (multiplikativa inverser).

5.1 Projektiv utvidgning

Låt oss beteckna denna utvidgning för $\mathbb{R}*$. Utvidgningen går ut på att vi lägger till ∞ som en egen punkt till $\mathbb{R}$ på ett sådant sätt att den reella tallinjen 'knyts ihop' och bildar en cirkel istället för en linje där ∞ binder ihop de två 'ändpunkterna' på den reella tallinjen. Den negativa och den positiva oändligheten bildar då en gemensam punkt.
Alltså

$$-\infty = \infty.$$

De aritmetiska operationerna förändras i utvidgningen på följande sätt:

$$x + \infty = \infty + x = \infty \quad om \quad x \neq \infty,$$

$$x \times \infty = \infty \times x = \infty \quad om \quad x \neq 0,$$

$$x/\infty = 0 \quad om \quad x \neq \infty,$$

$$x/0 = \infty \quad om \quad x \neq 0.$$[29]

Uttrycken $\infty/\infty, \infty \times 0$ och $\infty \pm \infty$ är samtliga odefinierade här.
För att ta ett konkret exempel (där nämnaren är ett intervall som innehåller 0) på hur utvidgningen fungerar så låt $\mathbf{a} = [3, 4]$ och $\mathbf{b} = [-2, 5]$, då får vi att $\mathbf{c} = \mathbf{a} \div \mathbf{b}$ beräknas på följande sätt:

$$[3, 4] \div [-2, 5] = [3, 4] \div ([-2, 0) \cup \{0\} \cup (0, 5]) =$$

$$([3, 4] \div [-2, 0)) \cup ([3, 4] \div 0) \cup ([3, 4] \div (0, 5]) =$$

[29][Tu11], s. 31-32

$$\{x \in \mathbb{R} : x \leq -\frac{3}{2}\} \cup \{\infty\} \cup \{x \in \mathbb{R} : \frac{3}{5} \leq x\}.$$

Eftersom den här typen av utvidgning kan ses som en cirkel snarare än en linje så kan vi alltså gå från ett större värde till ett mindre via oändligheten som då är både negativ och positiv oändlighet.

Genom den här utvidgningen kan man nu också representera division med 0. Exempelproblemet kan därmed skrivas ut som

$$[3,4] \div [-2,5] = \{x \in \mathbb{R} : x \leq -\tfrac{3}{2}\} \cup \{\infty\} \cup \{x \in \mathbb{R} : \tfrac{3}{5} \leq x\} = [\tfrac{3}{5}, -\tfrac{3}{2}].$$

För utvidgning av intervall skrivs mängden som

$$\mathbb{IR}* = \{[\underline{a}, \overline{a}] : \underline{a}, \overline{a} \in \mathbb{R}*\}$$

där utvidgningen existerar då $\underline{a} > \overline{a}$.

Problemet med projektiv utvidgning är dock att det ej går att jämföra element gällande ordning eftersom påståendet att $-\infty < +\infty$ ej är sant i projektiv utvidgning. Därav krävs en förbättrad utvidgning när vi arbetar med intervallanalys.

5.2 Affin utvidgning

Låt oss nu beteckna denna utvidgning av $\mathbb{R}$ som $\overline{\mathbb{R}}$.

Utvidgningen erhålles genom att till den reella mängden lägga till två skilda oändligheter; $-\infty$ och $+\infty$, vilket alltså ger oss att $\overline{\mathbb{IR}}$ kan definieras som alla intervallvärda element i det slutna intervallet $[-\infty, +\infty]$.

Affin utvidgning är oftast väldigt användbar inom analysen eftersom den, till skillnad från projektiv utvidgning, kan användas för att jämföra oändligheters storlekar. Den utvidgade aritmetiken till $\mathbb{IR}$ bildas på följande sätt:

$$-(+\infty) = -\infty,$$
$$-(-\infty) = +\infty,$$
$$x + (-\infty) = -\infty \quad om \quad x \neq +\infty,$$
$$x \times (\pm\infty) = \mp\infty \quad om \quad x < 0,$$
$$x + (+\infty) = +\infty \quad om \quad x \neq -\infty,$$
$$x \times (\pm) = \pm\infty \quad om \quad x > 0,$$
$$x/(\pm\infty) = 0 \quad om \quad x \neq \pm\infty.^{30}$$

Division med 0 samt $+\infty + -\infty$ är odefinierade i utvidgningen.

Utvidgningen av intervallmängden är här definierat som

$$\overline{\mathbb{IR}} = \{[\underline{a}, \overline{a}] : \underline{a}, \overline{a} \in \overline{\mathbb{R}}\}$$

där $\underline{a} \leq \overline{a}$ skrivs som ett vanligt intervall medan intervall med egenskapen $\underline{a} \geq \overline{a}$ kan låtas utskrivas $[u, v] = [-\infty, v] \cup [u, \infty]$ (där $v \leq u$ gäller).

Som tidigare nämndes så saknas unika reciprokaler, alltså multiplikativa inverser, för den affina utvidgningen $\overline{\mathbb{R}}$. Detta löses genom att lägga till så kallade betecknade nollor till utvidgningen: +0 och -0. På så sätt får samtliga element i $\overline{\mathbb{R}}$ unika reciprokaler (tidigare saknades detta för 0 och $\pm\infty$). Det uppstår då inga felmeddelanden eller programkraschar när man beräknar division med noll i datorn eftersom den odefinierade operationen 'division med noll' nu kan definieras på följande sätt:

$$\begin{aligned} 1/(+\infty) &= +0 \\ 1/(+0) &= +\infty \\ 1/(-\infty) &= -0 \\ 1/(-0) &= -\infty \end{aligned}$$ [31]

Utvidgningen fungerar även väl i IEEE:s standardsystem för flyttal eftersom 0 skrivs ut som vanligt så skillnaden syns endast i tecken-delen på flyttalet. Vid addition mellan olika betecknade nollor, inom IEEE:s standardformat, har man att

$$(+0) + (-0) = x - x = +0$$

utom vid avrundning mot $-\infty$ vilket ges från följande givna regler:

–För $x + x = x - (-x)$ så behåller x sitt tecken då x är 0.

–Vid nollsumma eller nolldifferens av två operander med olika tecken så skrivs nollsummans (eller nolldifferensens) tecken som + utom vid avrundningar mot $-\infty$ då den skrivs –. [32]

Detta är alltså vad som gäller för $\overline{\mathbb{R}}$. Vad gäller då för $\overline{\mathbb{IR}}$, det vill säga utvidgad intervalldivision?

[30] [Tu11], s. 32-33
[31] ibid, s. 33
[32] ibid, s. 33-34

Följande definierar utvidgad intervalldivision (för $\overline{\mathbb{IR}}$):
Låt återigen $\mathbf{a}$ och $\mathbf{b}$ beteckna intervallen $\mathbf{a} = [\underline{a}, \overline{a}]$ och $\mathbf{b} = [\underline{b}, \overline{b}]$. Då gäller följande:

$$1 \div \mathbf{b} \begin{cases} [1/\overline{b}, 1/\underline{b}] & \text{om} \quad 0 \notin \mathbf{b}, \\ [1/\overline{b}, +\infty] & \text{om} \quad \underline{b} = 0 < \overline{b}, \\ [-\infty, 1/\underline{b}] \cup [1/\overline{b}, +\infty] & \text{om} \quad \underline{b} < 0 < \overline{b}, \\ [-\infty, 1/\underline{b}] & \text{om} \quad \underline{b} < \overline{b} = 0 \\ \emptyset & \text{om} \quad \mathbf{b} = [0, 0]. \end{cases}$$

[33]

Fallen kan sedan utökas för att inkludera division med godtyckligt intervall som täljare:

$$\mathbf{a} \div \mathbf{b} = \begin{cases} \mathbf{a} \times [1/\overline{b}, 1/\underline{b}] & \text{om} \quad 0 \notin \mathbf{b}, \\ [-\infty, +\infty] & \text{om} \quad 0 \in \mathbf{a} \quad \text{och} \quad 0 \in \mathbf{b}, \\ [\overline{a}/\underline{b}, +\infty] & \text{om} \quad \overline{a} < 0 \quad \text{och} \quad \underline{b} < \overline{b} = 0, \\ [\overline{a}/\underline{b}, \overline{a}/\overline{b}] & \text{om} \quad \overline{a} < 0 \quad \text{och} \quad \underline{b} < 0 < \overline{b}, \\ [-\infty, \overline{a}/\overline{b}] & \text{om} \quad \overline{a} < 0 \quad \text{och} \quad 0 = \underline{b} < \overline{b}, \\ [-\infty, \underline{a}/\underline{b}] & \text{om} \quad 0 < \underline{a} \quad \text{och} \quad \underline{b} < \overline{b} = 0, \\ [\underline{a}/\overline{b}, \underline{a}/\underline{b}] & \text{om} \quad 0 < \underline{a} \quad \text{och} \quad \underline{b} < 0 < \overline{b}, \\ [\underline{a}/\overline{b}, +\infty] & \text{om} \quad 0 < \underline{a} \quad \text{och} \quad 0 = \underline{b} < \overline{b}, \\ \emptyset & \text{om} \quad 0 \notin \mathbf{a} \quad \text{och} \quad \mathbf{b} = [0, 0]. \end{cases}$$

[34]

Ovanstående generaliserar *Sats 1* då dessa kan bevisas vara inklusionsisotoniska. [35]

Programkoden för den utvidgning av intervalldivision som Moore tar upp i *Introduction to Interval Analysis* finns i avsnitt 8.2 som koden Xreciprocal.m.

5.3 Inkluderande mängder (csets)

De presenterade utvidgningarna räcker inte alltid till och kräver därför extra kompletteringar för att räkna med intervall som innehåller 0. Fall där de fallerar kan vara funktioner innehållandes exempelvis kvadratrötter:
Låt oss betrakta funktionen $f(x) = \sqrt{2x + (-3x^2)}$ över domänen $\mathbf{x} = [0, 1]$ vilken ger värdemängden $R(f; [0, 1]) = [0, 1/\sqrt{3}]$. För intervallfunktionen

[33] [Mo09], s. 110
[34] [Tu11], s. 34
[35] Beviset finns i Ratz *Inclusion isotone extended interval arithmetic – a toolbox update* (1996).

$F(\mathbf{x}) = \sqrt{2\mathbf{x} + (-3\mathbf{x}^2)}$ blir beräkningen emellertid en aningen annorlunda:

$$F([0,1]) = \sqrt{[2,2] \times [0,1] + [-3,-3] \times [0,1]^2} =$$

$$\sqrt{[2,2] \times [0,1] + [-3,-3] \times [0,1]} = \sqrt{[0,2] + [-3,0]} = \sqrt{[-3,2]}$$

Resultatet kommer bli komplext men om man istället använder sig av den naturliga domänen för funktionen, i det här fallet den reella $\mathbb{R}$, kan man på så sätt 'kapa' domänen innan operationen utförs på intervallet.

Detta benämns lös evaluering och kan definieras som att givet intervallet $\mathbf{x}$ så utförs

$$\sqrt{\mathbf{x}} = \sqrt{\mathbf{x} \cap [0, +\infty]}$$

vilket i det tidigare exemplet ger

$$\sqrt{[-3,2] \cap [0,+\infty]} = \sqrt{[0,2]} = [0, \sqrt{2}].$$

För att skapa ett system där man slipper undantag från regeln kan man, givet en reell funktion f: $D_f \implies \mathbb{R}$ där D_f är den största domän där f är väldefinierad och S är vår input, skapa inkluderande mängd-utvidgning (cset extension)

$$f* : \mathcal{P}\overline{\mathbb{R}} \implies \mathcal{P}\overline{\mathbb{R}}$$

genom *Definition 2*:

$$f*(S) = R(f; S \cap D_f) \cup \{\lim \zeta \to \zeta * f(\zeta) : \zeta \in D_f, \zeta* \in S \setminus D_f\}.^{36}$$

$\mathcal{P}\overline{\mathbb{R}}$ är här mängden av alla delmängder till den affina utvidgningen.

Det Definition 2 gör är att först begränsa S genom att ta snittet med den för funktionen naturliga domänen för att sedan tillämpa gränsvärden för att kontrollera eventuella öppna gränser.

Med de redskap som lagts fram kan nu en presentation av intervallanalysen göras.

[36] [Tu11], s. 35

6 Intervallanalys

Det uppstår en del problem när man använder klassisk analys på intervall eftersom vissa regler och definitioner är lösare i intervallaritmetiken, exempelvis sub-distribution.

Nedan följer en redogörelse av de viktigaste och mest grundläggande reglerna inom intervallanalysen.

6.1 Funktioner med intervall

Överlagringarna i Matlab för funktionerna som nämns i följande avsnitt finns i avsnitt 8.1.

Monotona funktioner är enkla att utvidga eftersom dessa är strikt växande eller sjunkande vilket medför att yttre gränserna för sökområdet utgör minimum- och maximumpunkter varför funktionerna då kan evalueras direkt vid ändpunkterna.

Generellt sett gäller följande för monotona funktioner:
Givet intervallet $\mathbf{a} = [\underline{a}, \overline{a}]$ och de godtyckliga elementen $\underline{b} \wedge \overline{b} \in \mathbf{a}$ där $\underline{b} \leq \overline{b}$ och där funktionen f sjunker eller ökar längs intervallet $\mathbf{a}$ så kommer antingen

$$f(\underline{b}) < f(\overline{b})$$

eller

$$f(\underline{b}) > f(\overline{b})$$

att vara sant. Värdemängden av $[\underline{b}, \overline{b}]$ kan då fås genom att använda funktionen f för $\underline{b}$ och $\overline{b}$ enskilt vilket ger att $f([\underline{b} \wedge \overline{b}]) = [\min f(\underline{b}), f(\overline{b}), \max f(\underline{b}), f(\overline{b})]$. Detta ger nedanstående regler för monotona funktioner med intervallet $\mathbf{x}$:

$$\begin{aligned} e^{\mathbf{x}} &= [e^{\underline{x}}, e^{\overline{x}}] \\ \sqrt{\mathbf{x}} &= [\sqrt{\underline{x}}, \sqrt{\overline{x}}], \qquad \text{om} \quad 0 \leq \underline{x} \\ log\mathbf{x} &= [\log \underline{x}, \log \overline{x}], \qquad \text{om} \quad 0 < \underline{x} \end{aligned}$$ [37]

För icke-monotona funktioner som har ändligt antal kända extrempunkter så gäller att följande regler kan ställas upp:

$$\mathbf{x}^n = \begin{cases} [1,1] & \text{om} \quad n = 0, \\ [1/\overline{x}, 1/\underline{x}]^{-n} & \text{om} \quad n \in \mathbb{Z}^- \quad \text{och} \quad 0 \notin x, \\ [\underline{x}^n, \overline{x}^n] & \text{om} \quad n \in \mathbb{Z}^+ \quad \text{är udda}, \\ [mig(x)^n, mag(x)^n] & \text{om} \quad n \in \mathbb{Z}^+ \quad \text{är jämn}. \end{cases}$$ [38]

[37] [Tu11], s. 47

Reglerna för $\mathbf{x}^n$ ger ett mindre intervall än för $\underbrace{\mathbf{x} \times ... \times \mathbf{x}}_{\text{n gånger}}$.

Detta medför att när funktioner utvidgas till intervallvärda funktioner så kan funktionernas värdemängder utvidgas till att bli mycket större än vad de behöver vara. Det som skapar skillnaderna i beteendet hos funktionerna är de skilda reglerna från vanlig aritmetik som till exempel det så kallade intervallberoendet (interval dependency), ett problem som uppstår då samma intervall upprepas oberoende av varandra flera gånger i en beräknings olika parametrar.

Problemet beror på att intervallaritmetiken ej skiljer på domän och variabel i funktionen vilket kan få det resulterande intervallet att skilja sig alltför mycket från vad som krävs för att uppnå den precision som man eftersöker. [39] [40] Ett exempel är den elementära funktionen

$$f(x) = x^2 - x$$

vars så kallade naturliga utvidgning (definieras som då alla x byts ut mot intervallet $\mathbf{x}$ rakt av i funktionen)

$$F(\mathbf{x}) = \mathbf{x}^2 - \mathbf{x}$$

som för domänen $[-1, 1]$ ger

$$[-1, 1]^2 - [-1, 1] = [0, 1] - [-1, 1] = [-1, 2]$$

trots att om vi istället försöker få så få upprepningar av $\mathbf{x}$ i funktionen som möjligt genom att förändra funktionen f till

$$f(x) = x^2 - x = (\mathbf{x} - \frac{1}{2})^2 - \frac{1}{4},$$

och då använda en annan naturlig utvidgning utifrån den, ger det mer precisa resultatet

$$([-1, 1] - \frac{1}{2})^2 - \frac{1}{4} = ([-1, 1] + [-\frac{1}{2}, -\frac{1}{2}])^2 - [\frac{1}{4}, \frac{1}{4}] =$$

$$([-\frac{3}{2}, \frac{1}{2}])^2 - [\frac{1}{4}, \frac{1}{4}] = [0, \frac{3}{2}] - [\frac{1}{4}, \frac{1}{4}] = [-\frac{1}{4}, \frac{5}{4}].$$

[38] [Tu11], s. 47
[39] [ibid
[40] [Mo09], s. 43-45, 66-67

Här får vi en begränsning av det resulterande intervallet eftersom vi nu har en variabel med en domän istället för flera variabler som upprepar samma domän i flera operationer. Då det mer precisa intervallet nu även är en skarpare inneslutning av resultatet än då man upprepar variabeln flera gånger så har man även att

$$\mathbf{x}^n \subseteq \underbrace{\mathbf{x} \times \mathbf{x} ... \times \mathbf{x}}_{\text{n gånger}} \quad [41]$$

Detta gäller dock endast strikt om vi istället för flera upprepningar av samma domän endast har en enda förekomst av denna i funktionen. Det kommer emellertid inte alltid gå att skapa skarpa utvidgningar så länge x uppkommer flera gånger i någon elementär funktion f.

Funktionen kan även användas för att visa sub-distribution:
I reell aritmetik så har vi att

$$x(x-1) = x^2 - x$$

men den sub-distributiva lagen säger istället att givet intervallet $\mathbf{x}$ så gäller

$$\mathbf{x}(\mathbf{x} - [1,1]) \subseteq \mathbf{x}^2 - \mathbf{x}.$$

För den tidigare domänen som användes för att visa intervallberoende och funktionerna

$$H(\mathbf{x}) = \mathbf{x}(\mathbf{x} - 1)$$

samt

$$F(\mathbf{x}) = \mathbf{x}^2 - \mathbf{x}$$

så har vi att

$$H([-1,1]) = [-1,1] \times ([-1,1] - [1,1]) = [-1,1] \times [-2,0] = [-2,2]$$

och

$$F([-1,1]) = [-1,1]^2 - [-1,1] = [0,1] - [-1,1] = [-1,2].$$

Här ser vi alltså tydligt att

$$F([-1,1] \neq H([-1,1])$$

utan snarare gäller

$$F([-1,1]) \subseteq H([-1,1]),$$

alltså sub-distribution.

[41] [Tu11], s. 47

Trigonometriska funktioner är en annan typ av funktioner som är relativt lätta att utvidga över intervall:
Om $S^+ = \{2k\pi + \pi/2 : k \in \mathbb{Z}\}$ och $S^- = \{2k\pi - \pi/2 : k \in \mathbb{Z}\}$ krävs bara att man vet huruvida intervallet $\mathbf{x}$ skär S^+ eller S^- vilket sedan används för följande regler:

$$\sin \mathbf{x} = \begin{cases} [-1,1] & : \text{om} \quad \mathbf{x} \cap S^- \neq [\emptyset] \quad \text{och} \quad \mathbf{x} \cap S^+ \neq [\emptyset] \\ [-1, \max\{\sin \underline{x}, \sin \overline{x}\}] & : \text{om} \quad \mathbf{x} \cap S^- \neq [\emptyset] \quad \text{och} \quad \mathbf{x} \cap S^+ = [\emptyset] \\ [\min\{\sin \underline{x}, \sin \overline{x}\}, 1] & : \text{om} \quad \mathbf{x} \cap S^- = [\emptyset] \quad \text{och} \quad \mathbf{x} \cap S^+ \neq [\emptyset] \\ [\min\{\sin \underline{x}, \sin \overline{x}\}, \max\{\sin \underline{x}, \sin \overline{x}\}] & : \text{om} \quad \mathbf{x} \cap S^- = [\emptyset] \quad \text{och} \quad \mathbf{x} \cap S^+ = [\emptyset] \end{cases}$$

För övriga trigonometriska funktioner kan standardidentiteter användas för att på så sätt alltid kunna räkna i sinus, exempelvis genom identiteten

$$\cos \mathbf{x} = \sin(\mathbf{x} + \frac{\pi}{2}).$$

Är $rad(\mathbf{x}) \geq \pi$ kan man direkt konstatera att både S^+ och S^- har icketomma gemensamma mängder med $\mathbf{x}$ eftersom om $a \in S^+$ och $b \in S^-$ så är $a - b \geq \pi$.

Sinus-funktionen i Matlab behöver även den överlagras för att kunna hantera intervall och stämma överens med de regler vi satt upp för sinusfunktionen över $\mathbb{IR}$. Funktionen kan överlagras med Matlab-filen som beskrivs i avsnitt 8.1 som Sin.m.

Elementära funktioner, som är uppbyggda av standardfunktionerna, har alltid oändligt många olika utvidgningar då de utvidgas för intervall men har då även en naturlig utvidgning där alla variabler är utbytta mot intervallvärda variabler rakt av.
Ett exempel på detta är

$$f(x) = x^3 + x$$

vars naturliga intervallutvidgning då är

$$F(\mathbf{x}) = \mathbf{x}^3 + \mathbf{x}$$

och alltså inte exempelvis

$$F(\mathbf{x}) = \mathbf{x}(\mathbf{x}^2 + [1,1]).$$

Utvidgningarna av elementära funktioner till $\mathbb{IR}$ är dessutom inklusionsisotoniska när de är väldefinierade.
Följande definition gäller:

Definition 3
Låt $\mathbf{x} \in \mathbb{IR}$.
En intervallvärd funktion $F : \mathbf{x} \cap \mathbb{IR} \implies \mathbb{IR}$ är inklusionsisotonisk om man, för alla $\mathbf{z} \subseteq \mathbf{z}' \subseteq \mathbf{x}$, har att $F(\mathbf{z}) \subseteq F(\mathbf{z}')$.
Definitionen utökas sedan till

Sats 2: Intervallanalysens fundamentalsats
Givet en elementär funktion f och en naturlig utvidgning F så att $F(\mathbf{x})$ är väldefinierad för något $\mathbf{x} \in \mathbb{IR}$ så har vi att

$$(1) \qquad \mathbf{z} \subseteq \mathbf{z}' \subseteq \mathbf{x} \implies F(\mathbf{z}) \subseteq F(\mathbf{z}'),$$

$$(2) \qquad R(f;\mathbf{x}) \subseteq F(\mathbf{x}),$$ [42] [43]

där (1) beskriver inklusionsistoniken och (2) avgränsning av värdemängden.[44]

Satsen är väldigt viktig då vi utifrån den kan konstatera att om ett element y ligger i $R(f;\mathbf{x})$ så måste den även ligga i $F(\mathbf{x})$ vilket alltså även betyder att givet ett element y så

$$y \notin F(\mathbf{x}) \implies y \notin R(f;\mathbf{x}).$$

Det här kan senare användas för att exempelvis begränsa antalet möjliga mängder som innehåller rötter till en funktion så länge vi vet att $0 \notin F(\mathbf{x})$ eftersom den då ej heller kan finnas i $R(f;\mathbf{x})$ och så fortsätter vi på samma sätt med en ny domän. Om problem uppstår, exempelvis om $F(\mathbf{x})$ ej är väldefinierad, så kan (1) användas för att undkomma problemet.

Exempel på då (2) inte räcker till utan även kräver (1):

Finn en förslutning av $R(f;[0,4])$ där $f(x) = \sqrt{x + \sin(x)}$.
Vi har då att

$$R(f;[0,4]) \subseteq F([0,4]) = \sqrt{[0,4] + \sin([0,4])} = \sqrt{[0,4] + [\sin 4, 1]} = \sqrt{\sin 4, 5]}$$

vilket ej kan beräknas på grund av att $\sin(4)$ är negativt. Vi tillför då (1) och delar vårt intervall:

$$[0,2] = [0,1] \cup [1,2].$$

[43] [Tu11], s. 49-50
[43] [Mo09], s. 47
[44] Fullständigt bevis finns i [Tu11], s. 50 och [Mo09], s. 47

Av inklusionsisotoniken får vi då att

$$R(f;[0,4]) = R(f;[0,2]) \cup R(f;[2,4]) \subseteq F([0,2]) \cup F([2,4]) =$$

$$\sqrt{[0,2] + \sin([0,2])} \cup \sqrt{[2,4] + \sin([2,4])} =$$

$$\sqrt{[0,2] + [\sin 0, \sin \frac{\pi}{2}]} \cup \sqrt{[2,4] + [\sin 4, \sin 2]} =$$

$$\sqrt{[0,2] + [0,1]} \cup \sqrt{[2 + \sin 4, 4 + \sin 2]} =$$

$$[0, \sqrt{3}] \cup [\sqrt{2 + \sin 4}, \sqrt{4 + \sin 2}] = [0, \sqrt{4 + \sin 2}] \subseteq [0, 2.2157]$$

vilket inte ger oss någon skarp men en för uppgiften valid inneslutning. Genom att dela upp intervallet $\mathbf{x}$ i mindre och mindre segment och därefter beräkna F för varje enskilt segment så kan unionen av resultaten ge ett godtyckligt mycket mindre gällande överskattningen av det verkliga resultatet $R(f;\mathbf{x})$.

6.2 Lipschitz

Man kan se det föregående exemplet som att man delar in funktionen i mindre och mindre sektioner tills man har något beräkningsbart och sedan tar unionen mellan dessa bitar. Ett problem som man emellertid vill undvika är de delar av funktioners värdemängder som går mot oändligheten eftersom 'inrutandet' här blir väldigt problematiskt att genomföra. Detta löses genom att kontrollera om funktionen är en så kallad Lipschitz-funktion vilket bestäms genom följande definition:

Definition 4
En funktion $f : D \implies \mathbb{R}$ är Lipschitz om det existerar en positiv konstant K så att vi har, för alla $x, y \in D$, att $|f(x) - f(y)| \leq K|x - y|$. K är här Lipschitz-konstanten för f över D.

I definitionen ser vi att

$$|f(x) - f(y)| \leq K|x - y|$$

vilket även kan skrivas som

$$\frac{|f(x) - f(y)|}{|x - y|} \leq K. \quad [45] \quad [46]$$

Alltså kan vi se lösningen som att ta reda på funktionen f:s derivata och sedan sätta K som derivatans maximala värde.

Om $\mathfrak{E}$ är mängden av alla elementära funktioner så är $\mathfrak{E}_{\mathfrak{L}}$ mängden av element från $\mathfrak{E}$ vars alla deluttryck är Lipschitz: $\mathfrak{E}_{\mathfrak{L}} = \{f \in \mathfrak{E} :$ varje deluttryck av f är Lipschitz $\}$.
För att utnyttja Lipschitz-funktionerna så kan vi använda oss utav följande sats:

Sats 3:
Låt $f : I \implies \mathbb{R}$ med $f \in \mathfrak{E}_{\mathfrak{L}}$, och låt därefter F vara en inklusionisotonisk intervallutvidgning av f på så sätt att $F(\mathbf{x})$ är väldefinierat för något $\mathbf{x} \subseteq I$. Då följer att det existerar ett positivt reellt tal K, som beror på F och $\mathbf{x}$, så att om

$$\mathbf{x} = \cup_{i=1}^{k} \mathbf{x}^i$$

så gäller

$$R(f;\mathbf{x}) \subseteq \bigcup_{i=1}^{k} F(\mathbf{x}^i) \subseteq F(\mathbf{x})$$

och

$$rad(\bigcup_{i=1}^{k} F(\mathbf{x}^i)) \leq rad(R(f;\mathbf{x})) + K \max_{i=1,\ldots,k} rad(\mathbf{x}^i).$$ [47] [48]

För att förklara det i ord säger satsen att om vi delar upp ett intervall i delintervall och tar unionen av dessa så vet vi att unionen är en delmängd av intervallfunktionen och att $R(f;\mathbf{x})$ är en delmängd av unionen av delintervallen. Dessutom är radien av delintervallens union mindre än eller lika med radien för

$$R(f;\mathbf{x}) + K \max_{i=1,\ldots,k} rad(\mathbf{x}^i)$$

där just

$$K \max_{i=1,\ldots,k} rad(\mathbf{x}^i)$$

är den längden som skiljer unionen av delintervallen från $R(f;\mathbf{x})$ i radie. Vi vet detta eftersom om K är maximala derivatan av f så måste ju alltså funktionen f ligga innanför 'en box' med f:s minimala och maximala derivata som begränsningar eftersom funktionen f aldrig kommer att befinna sig utanför sin maximala och minimala lutning.

[46] [Tu11], s 52
[46] [Mo09], s. 53
[48] [Tu11], s. 53
[48] Fullständigt bevis finns i [Tu11], s. 53-54

Del två av satsen kan i princip sammanfattas med att, om de listade kraven i början av satsen gäller, så går överskattning mot noll linjärt minst lika snabbt som domänen krymper:

$$rad(\mathbf{x}) = \mathcal{O}(\epsilon) \implies d(R(f;\mathbf{x}), F(\mathbf{x})) = \mathcal{O}(\epsilon)$$

där $d(\mathbf{a}, \mathbf{b})$ definieras som Haussdorf-distansen mellan $\mathbf{a}$ och $\mathbf{b}$ och $\mathcal{O}(\epsilon)$ alltså är den maximala överskattningen av $R(f;\mathbf{x})$ gänser:

$$d(\mathbf{a}, \mathbf{b}) = \max\{|\underline{a} - \underline{b}|, \overline{a} - \overline{b}\}.$$

Då just Lipschitz-funktioner uppfyller att

$$rad(R(f;\mathbf{x})) = \mathcal{O}(rad(\mathbf{x}))$$

så gäller det att

$$rad(\mathbf{x}) = \mathcal{O}(\epsilon) \implies rad(F(\mathbf{x})) = \mathcal{O}(\epsilon)$$

vilket innebär att bredden av inneslutningen beskärs minst linjärt med ϵ. Därmed kan vi även bestämma hur mycket vi då överskattar funktionen med intervallfunktionen F eftersom detta då är avståndet till funktionen f:s maximum- och minimumpunkter (om en derivata går att finna för f). Lipschitz-funktioner kan på så sätt användas för att dela upp ett sökområde i så pass små intervall att överapproximeringen av värdemängderna blir så liten man vill ha den.

6.3 Derivata

För att derivera funktioner som involverar intervall deriverar man först funktionen f vilket ger f 'och utvidgar sedan denna till intervallfunktionen
F '. Alltså har vi exempelvis att derivatan av $F(\mathbf{x}) = \mathbf{x}^2$ blir detsamma som att först derivera $f(x) = x^2$ vilket blir $f'(x) = 2x$ och sedan utvidga funktionen till $F'(\mathbf{x}) = 2\mathbf{x}$.

7 Metoder

För att på ett systematiskt sätt kunna ta reda på huruvida en intervallvärd värdemängd innehåller nollor behövs metoder som garanterar att vi håller oss innanför de ramar som tidigare satts upp. En av de enklare metoderna för att hitta nollor i en funktions domän är bisektionsmetoden.

7.1 Bisektionsmetoden

För bisektionsmetoden arbetar man med intervall vilket förenklar det hela en aning i vårt fall.

Börja med att genom Bolzanos sats[49] låta en känd funktions värde inom ett intervall någonstans vara lika med noll då funktionen byter tecken inom intervallet. Vi betecknar intervallet som [a, b] och tar sedan en ny punkt som ligger som mittpunkt i det föregående använda intervallet. Låt oss kalla denna punkt för

$$c = \frac{a+b}{2}.$$

Om c inte är ett nollställe till funktionen f kommer f(c) att ha antingen samma tecken som f(a) eller f(b). Den som har det av a eller b ersätter man med c och bildar då ett nytt mindre intervall.

Låt oss säga att f(b) hade samma tecken som f(c), då får vi det nya intervallet [a, c]. Man utför sedan flera iterationer tills man fått den noggrannhet man vill ha eftersom intervallet kommer att konvergera mot det nollställe man söker.

För själva lösningen vet vi att skillnaden mellan den n:te iterationens värde, c_n och lösningen c är begränsat av

$$|c_n - c| \leq \frac{|b-a|}{2^n},$$

vilket alltså kan användas för att avgöra hur många iterationer som krävs för att konvergera mot lösningen om vi vill uppnå ett resultat inom en viss feltolerans.

För intervall är det bara att använda den naturliga utvidgningen för funktionen F, lägga in sitt sökområde $\mathbf{x}$ och sedan köra bisektionen som ovan: sökområdet delas upp i två intervall som sedan varförsig stoppas in i funktionen F så att gränsernas tecken kan kontrolleras. Om båda gränser har samma tecken för ett intervall innebär det att ett nollställe ej existerar i sökområdet

[49] Om c är ett tal mellan f(a) och f(b) för en kontinuerlig funktion f, på ett slutet och begränsat intervall, så finns det ett värde x_c som motsvarar c mellan a och b med egenskapen $c = f(x_c)$

(utifrån Intervallanalysens fundementalsats) vilket medför att det förkastas. Ett sökområde som innehåller ett nollställe delas även det i flera bitar varefter varje bit körs som en ny input i bisektionsmetoden för att kontrollera efter flera nollställen. Varje nollställe som hittas sparas sedan i en lista som sedan skickas ut som resultat efter uppnådd tolerans.

7.2 Newtons metod (Newton-Raphsons metod)

Metoden tillhör de så kallade Householdersmetoderna som är en uppsättning algoritmer vilka används för att hitta rötter till envariabla reella funktioner med kontinuerliga derivator upp till någon ordning d+1. Just Newton-Raphsons metod är en Householdersmetod av ordning 1 (det vill säga där d = 1).[50] Funktionerna konvergerar olika fort beroende på metodens ordning.

Exempelvis tar Newton-Raphsons metod längre tid att konvergera än en Householdersmetod av ordning 2. Emellertid blir det svårare att beräkna högre ordningar av Householdersmetoden vilket man ser på metodens allmänna form (ju högre ordning på metoden desto högre ordning på derivatan måste beräknas):

$$x_{n+1} = x_n + d\frac{(1/f)^{(d-1)}(x_n)}{(1/f)^d(x_n)}$$

Newton-Raphsons metod är då av ordning 1 vilket ger d = 0 och medför då den allmänna formen

$$x_{n+1} = x_n + 1\frac{(1/f)(x_n)}{(1/f)^1(x_n)} = x_n + 1\frac{1}{f(x_n)} \times \left(\frac{-f'(x_n)}{f(x_n)^2}\right)^{-1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

där f är en kontinuerlig differentierbar funktion. Alltså är Newton-Raphsons metod i sin vanliga form för reella värden:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Nu måste vi emellertid använda intervall i beräkningarna vilket medför att formen behöver modifieras en del. Om vi utgår från medelvärdessatsen vet vi att

$$f(x) = f(y) + f'(s)(x - y)$$

för något s mellan de reella värdena x och y. Låter vi då $[\underline{a}, \overline{a}]$ vara ett intervall där en lösning för $f(x) = 0$ söks så kommer lösningen att uppfylla

$$f(y) + f'(s)(x - y) = 0$$

[50] Gourdon, Xavier and Sebah, Pascal, http://numbers.computation.free.fr/Constants/Algorithms/newton.html, senast kontrollerad 18-06-2014

för något $y \in [\underline{a}, \overline{a}]$, särskilt

$$y = mid([\underline{a}, \overline{a}]) = \frac{\underline{a} + \overline{a}}{2}.$$

Det här kan då skrivas om så att vi slutligen får

$$x = y - \frac{f(y)}{f'(s)}.$$

Vi får en sats som hanterar intervall med Newtons metod om givet algoritmen

$$\mathbf{x}_{k+1} = \mathbf{x}_k \cap N(\mathbf{x}_k) \qquad (k \in \mathbb{N}),$$

där N kallas Newtonoperatorn och definieras som

$$N(\mathbf{x}) = m(\mathbf{x}) - \frac{f(m(\mathbf{x}))}{F'(\mathbf{x})},$$

så låter vi $F'(\mathbf{x})$ vara en inklusionsisoton intervallutvidgning av $f'(x)$. Från tidigare har vi då att x är ett element i $N(\mathbf{x})$ om $y = m(\mathbf{x})$.
Om $x \in \mathbf{x}$ så kommer $s \in \mathbf{x}$ vilket medför att $x \in \mathbf{x}_k$ för alla $k \in \mathbb{N}$ om det är en del av $\mathbf{x}_0$, det vill säga så länge $\mathbf{x}_k \subseteq \mathbf{x}_0$.

Följande sats kan nu erhållas ur detta:

Sats 4:
Om ett intervall $\mathbf{x}_0$ innehåller en nollpunkt, x_z, för f(x) så gör även $\mathbf{x}_k$ det för alla $k \in \mathbb{N}$ vilket även definieras ovan som

$$\mathbf{x}_{k+1} = \mathbf{x}_k \cap N(\mathbf{x}_k) \qquad (k \in \mathbb{N}),$$

där $N(\mathbf{x}_0)$ är väldefinierad.

Intervallen $\mathbf{x}_{(k)}$ bildar dessutom en inkapslande sekvens som konvergerar mot x_z om $0 \notin F'(\mathbf{x}_0)$.[51]

För att bevisa existens av nollställen i ett sökområde samt finna ett stoppvillkor för då precis ett nollställe finns i sökområdet så är Newtonoperatorn ett oerhört kraftfullt verktyg. Utifrån följande sats fås redskapen att bygga en programkod för en välfungerande inervallbaserad Newton-Raphson-metod i Matlab:

[51]Bevis och sats finns i [Mo09], s. 106

Sats 5:
Låt $f : \mathbf{x} \to \mathbb{R}$ vara kontinuerlig och deriverbar två gånger. Antag därefter att $N(\mathbf{x})$ är väldefinierad för ett godtyckligt $\mathbf{x} \in \mathbb{IR}$. Då gäller

$$a) \quad \text{om} \quad N(\mathbf{x}) \cap \mathbf{x} = \emptyset, \quad \text{så innehåller} \quad \mathbf{x} \quad \text{inga nollställen till f;}$$

$$b) \quad \text{om} \quad N(\mathbf{x}) \subseteq \mathbf{x}, \quad \text{så innehåller} \quad \mathbf{x} \quad \text{exakt ett nollställe till f.}^{52}$$

Programkoden för metoden kan då göras genom att man kör metoden men delar sökområdet i delintervall ända tills b) från sats 5 uppfylls varefter det givna intervallet förs in på en lista efter att det uppnått efterfrågad tolerans. Alla delintervall som uppfyller a) förkastas helt. Programkoden finns i avsnitt 8.2 som intervalNewtonSearch (skriven av Warwick Tucker).

Man kan visa existensen av flera nollställen på en gång genom att kombinera bisektionsmetoden med Newton-Raphson-metoden och då utföra Newton-Raphson-metoden när $0 \notin F'(\mathbf{x})$ så att man undviker division med noll. I övriga fall utförs bisektion. Koden för detta finns i sektion 8.2 som Bisnew.m. Koden för en intervallbaserad Newton-Raphson-metoden finns även den i sektion 8.2. Problemet med den här typen av kombinerad metod är dock att den inte kan garantera bevisad existens av samtliga nollställen. För att hitta alla nollställen krävs istället att man utvidgar intervalldivisionen genom proceduren som visades i sektion 5.2. Detta kan till exempel göras genom att tillsätta xreciprocal.m till intervalNewtonSearch.m (båda finns i sektion 8.2). Det man behöver tänka på då är att ett av fallen ger en uppdelning av det itererade sökområdet i två delar så att man får köra varje delintervall för sig i metoden igen. Den utvidgade metoden garanterar sedan, utifrån sats 4 och 5, att man finner alla nollställen till den inmatade funktionen.

[52] Bevis och sats finns i [Tu11], s. 79-80

8 Programkoder, analys

Nedan följer de senare programkoderna från avsnitt 6.1 samt 7.2.

8.1 Funktioner

De beskrivna funktionerna i avsnitt 6.1 överlagrades till intervall-klassen på följande sätt:

Log.m (logaritm-funktionen)

```
function result  = log(a)
%Överlagring av log-funktionen s.a. den fungerar för intervall
[a]=cast(a, interval);
if a.lo>0
setround(-inf);
lo=log(a.lo);
setround(+inf);
hi=log(a.hi);
setround(0.5);
result=interval(lo, hi);
else
    disp('error, komplext')
end

end
```

Sqrt.m (funktionen för kvadratroten ur)

```
function result  = sqrt(a)
%Överlagring av sqrt-funktionen s.a. den fungerar för ...
   intervall
[a]=cast(a, interval);
if a.lo>=0
setround(-inf);
lo=sqrt(a.lo);
setround(+inf);
hi=sqrt(a.hi);
setround(0.5);
result=interval(lo, hi);
else
    disp('error, komplext')
end

end
```

Exp.m (e^x-funktionen)

```
function result  = exp(a)
%Överlagring av exp-funktionen s.a. den fungerar för intervall
[a]=cast(a, interval);
setround(-inf);
lo=exp(a.lo);
setround(+inf);
hi=exp(a.hi);
setround(0.5);
result=interval(lo, hi);

end
```

Mpower.m (potens-funktionen)

```
function result  = mpower(a,n)
%Överlagring av mpower-funktionen s.a. den fungerar för ...
   intervall
[a]=cast(a, interval);
if mod(n, 2)==1
setround(-inf);
lo=mpower(a.lo,n);
setround(+inf);
hi=mpower(a.hi,n);
setround(0.5);
result=interval(lo, hi);
elseif mod(n, 2)==0
setround(-inf);
lo=mig(a)^n;
setround(+inf);
hi=mag(a)^n;
setround(0.5);
result=interval(lo, hi);
elseif n<=0
setround(-inf);
lo=1/a.hi;
setround(+inf);
hi=1/a.lo;
setround(0.5);
result=interval(lo, hi)^(-n);
else
    result=interval(1,1);

end

end
```

Sin.m (sinus-funktionen)

```
function result = sin(a)
%Överlagring av sinus-funktionen så att den körs ...
   intervallaritmetiskt.
[a]=cast(a, interval);
if (abs(a.hi - a.lo) >= 2*pi)
    %Om diametern är större än eller lika med 2*pi så ...
       träffar intervallet
    %både S+ och S- (S+ och S- finns på sida 47 i Tuckers)

    result = interval(-1, 1);

else
  if (a.lo <= 0) && (a.hi >= 0) %VILLKOR 1
      %Eftersom intervallet a här ej har större diameter ...
         än 2*pi och lägre
      %gräns är negativ, övre gräns positiv så kommer ...
         intervallet ligga som
      %lägst mellan precis innan -2*pi och 0 och som högst ...
         mellan 0 och
      %precis innan 2*pi. Alltså kan det bara träffa ...
         -3*pi/2, -pi/2, pi/2
      %och 3*pi/2. Det är 2*pi mellan 3*pi/2 och -pi/2 ...
         (även för respektive
      %teckenbyte). Dock kan intervallet träffa -pi/2 och ...
         pi/2 samtidigt,
      %därför krävs subvillkoren under varje if:
      x=a.lo;
      y=a.hi;


      if ((a.lo <= (pi/2)) && (a.hi >= (pi/2)))||((a.lo <= ...
         (-3*pi/2)) && (a.hi >= (-3*pi/2))) %villkor alfa
          if ((a.lo <= (-pi/2))&& (a.hi >= (pi/2))) ...
             %subvillkor för alfa
              result=interval(-1,1);
          else
              result = interval(min([sin(x) sin(y)]), 1);
          end
      elseif ((a.lo <= (3*pi/2)) && (a.hi >= (3*pi/2)))|| ...
         ((a.lo <= (-pi/2)) && (a.hi >= (-pi/2))) ...
         %villkor beta
          if ((a.lo <= (-pi/2))&& (a.hi >= ...
             (pi/2)))%subvillkor för beta
              result=interval(-1,1);
          else
              result = interval(-1, max([sin(x) sin(y)]));
```

```
34            end
35        else ...

            %villkor gamma
36            result = interval(min([sin(x) sin(y)]), ...
                  max([sin(x) sin(y)]));
37        end
38
39    elseif (a.lo < 0) && (a.hi < 0)%VILLKOR 2
40         x=a.lo;
41          y=a.hi;
42          diameter=abs(y-x);
43
44      a.hi = mod(a.hi, 2*pi);
45      %Övre gränsen beskärs med så många 2*pi som möjligt ...
           tills den ligger "nära" 0.
46      a.lo=a.hi-diameter;
47      %Eftersom det finns en möjlighet att ett lägre ...
           negativt värde beskärs
48      %fler ggr med 2*pi än det övre gränsvärdet innan det ...
           blir positivt och
49      %tillräckligt nära 0 så måste den lägre gränsen fås ...
           genom att skillnad
50      %mellan gränserna dras av från den beskärda övre ...
           gränsen istället för
51      %att modulu(lägre gräns, 2*pi) utförs.
52      %Övre gräns kan även här hamna väldigt nära 2*pi då ...
           det beskärs men nedre gräns kan gå så lågt som ...
           till precis ovanför -2*pi
53      %(om övre skulle vara något som ger en rest lika med 0 ...
           då övre gräns
54      %blir 0.
55
56
57      if ((a.lo <= (pi/2)) && (a.hi >= (pi/2)))||((a.lo <= ...
           (-3*pi/2)) && (a.hi >= (-3*pi/2))) %villkor alfa
58            if ((a.lo <= (-pi/2))&& (a.hi >= (pi/2))) ...
                  %subvillkor för alfa
59                result=interval(-1,1);
60            else
61                result = interval(min([sin(x) sin(y)]), 1);
62            end
63        elseif ((a.lo <= (3*pi/2)) && (a.hi >= (3*pi/2)))|| ...
              ((a.lo <= (-pi/2)) && (a.hi >= (-pi/2))) ...
              %villkor beta
64            if ((a.lo <= (-pi/2))&& (a.hi >= ...
                  (pi/2)))%subvillkor för beta
65                result=interval(-1,1);
66            else
```

```
67                result = interval(-1, max([sin(x) sin(y)]));
68             end
69          else ...

               %villkor gamma
70             result = interval(min([sin(x) sin(y)]), ...
                  max([sin(x) sin(y)]));
71       end
72
73    else                          %VILLKOR 3
74           x=a.lo;
75          y=a.hi;
76     diameter=abs(y-x);
77       a.hi = mod(a.hi, 2*pi);
78       a.lo=a.hi-diameter;
79
80
81        if ((a.lo <= (pi/2)) && (a.hi >= (pi/2)))||((a.lo <= ...
             (-3*pi/2)) && (a.hi >= (-3*pi/2))) %villkor alfa
82             if ((a.lo <= (-pi/2))&& (a.hi >= (pi/2))) ...
                  %subvillkor för alfa
83                result=interval(-1,1);
84             else
85                result = interval(min([sin(x) sin(y)]), 1);
86             end
87          elseif ((a.lo <= (3*pi/2)) && (a.hi >= (3*pi/2)))|| ...
               ((a.lo <= (-pi/2)) && (a.hi >= (-pi/2))) ...
               %villkor beta
88             if ((a.lo <= (-pi/2))&& (a.hi >= ...
                  (pi/2)))%subvillkor för beta
89                result=interval(-1,1);
90             else
91                result = interval(-1, max([sin(x) sin(y)]));
92             end
93          else ...

               %villkor gamma
94             result = interval(min([sin(x) sin(y)]), ...
                  max([sin(x) sin(y)]));
95       end
96    end %slut
97
98
99 end
100
101
102 end
```

Cos.m (Cosinus-funktionen)

```
function result = cos( a )
%Överlagra cos
[a]=cast(a,interval);
result=sin(a+(pi/2));


end
```

8.2 Metod

Istället för $1 \div \mathbf{x}$ kommer filen xreciprocal.m att användas i Newton-Raphson-koden. Detta på grund av att ett av fallen i den utvidgade division som hanterar division med noll innehåller unionen av flera intervall. Funktionen tar även upp de övriga fallen i Moores utvidgade division.

Xreciprocal.m [53]

```
function [y1, y2, two] = xreciprocal(x)
%xreciprocal används då 1/x innehåller x som ett intervall ...
    med 0, dvs. då
%division med noll föreligger. Oftast är inte det här ...
    intervallet ett
%intervall över x utan snarare intervallet över y, typ ...
    f'(x) med
%intervallet x, därav matas alltså f'(x) in som input, ...
    inte x.
% Följer standarden från Moore's Introduction to interval ...
    analysis (2009)
% sid. 112.

%Obs! Fallet c = d = 0 ingår inte här! Dvs. ej division ...
    med det
%degenererade intervallet [0, 0].

if (x.lo > 0) ||(x.hi < 0) %då körs vanlig intervalldivision
    two = 0;
    y1 = 1/x;
    y2 = interval();
elseif (x.lo == 0) && (x.hi > 0)
    %Fall 1: Om 0 ingår i [c,d] och c = 0 < d så är ...
       1/[c,d]=[1/d, +Inf). Se
```

[53] [Mo09], s. 112

```
18      %sid. 110 i Moore
19      two=0;
20      lowbound=interval(1,1)/interval(x.hi, x.hi);
21      y1=interval(lowbound.lo, Inf);
22      y2=interval();
23  elseif (x.lo<0) && (x.hi>0)
24      %Fall 2: Om c < 0 < d så är 1/[c,d] = ...
            (-Inf,1/c]union[1/d, +Inf), sid. 110
25     two=1;
26     upbound=interval(1,1)/interval(x.lo, x.lo);
27     y1=interval(-Inf, upbound.hi);
28     lowbound=interval(1,1)/interval(x.hi, x.hi);
29     y2=interval(lowbound.lo, Inf);
30  elseif (x.lo<0)&&(x.hi == 0)
31      %Fall 3: Om c < d = 0 så är 1/[c,d] = (-Inf, 1/c)
32     two=0;
33     upbound=interval(1,1)/interval(x.lo, x.lo);
34     y1=interval(-Inf, upbound.hi);
35     y2=interval();
36  else %Om x=0, fallet ej med i texten på sid. 110.
37      two=1;
38      y1=interval();
39      y2=interval();
40  end
```

Nedan följer bisektionsmetoden anpassad för intervall.
Bisektion.m:

```
1  function result = bisektion(f, x, tol)
2
3  k=f(x);
4
5  if (k.lo<=0)&&(k.hi>=0)
6      if Diam(x)<=tol
7
8          result = x
9
10     else
11         a=interval(x.lo, mid(x));
12         b=interval(mid(x), x.hi);
13         bisektion(f, a, tol);
14         bisektion(f, b, tol);
15     end
16 end
17 end
```

Följande kod är hämtad från [Tu11] och utför Newton-Raphson-metoden anpassad för intervall men dock utan utvidgad intervalldivision, det vill säga

den är ej anpassad för att kunna dividera med intervall innehållandes 0:

intervalNewtonSearch [54]:

```
function x=intervalNewtonSearch(f, fp, x, tol)
x0=x;
printInterval(x);
while (Diam(x)>tol)
    x=newtonStep(f, fp, x);
    if isempty(x)
        disp('Inga nollor i sökintervallet');
        return;
    end
    printInterval(x);
end
if (x<x0)
    fprintf('Finns unikt nollställe i slutintervallet');
end
end

function y=newtonStep(f, fp, x)
mx=interval(mid(x));
Nx=mx-(f(mx)/fp(x));
y=x & Nx;
end

function printInterval(x)
fprintf('x=[%.17f, %.17f]; rad=%e\n', x.lo, x.hi, Diam(x)/2);
end
```

intervalNewtonSearch kan sedan användas tillsammans med bisektionsmetoden för att hitta flera nollställen. Detta genom att bisektion genomförs om 0 finns i värdemängden för den inmatade funktionens derivata medan Newton-Raphson används då detta ej är fallet.

[54] [Tu11], s. 80

Bisnew.m :

```
function result = bisnew(f, fp, x, tol)

g=fp(x);
k=f(x);


if (k.lo<=0)&&(k.hi>=0)
    if (g.lo>0)||(g.hi<0)
        %Om 0 ej existerar i värdemängden hos derivatan av ...
           den inmatade
        % funktionen så körs Newton-Raphson-metoden.
        intervalNewtonSearch(f, fp, x, tol);
    else %Annars utförs intervallanpassad bisektion.
        if Diam(x)<=tol

            result = x

        else
            a=interval(x.lo, mid(x));
            b=interval(mid(x), x.hi);
            bisnew(f, fp, a, tol);
            bisnew(f, fp, b, tol);
        end
    end
end
end
```

Körs bisnew.m sedan med funktionen f(x)=sin(cos(x-3)), sökintervallet [-10, 10] samt toleransen 0.001 så erhålls följande sex stycken intervall där nollställen bevisligen existerar[55]

```
x=[-7.99601345274812840, -7.99553519977762670]; ...
    rad=2.391265e-04
x=[-4.85398163463337970, -4.85398163392099140]; ...
    rad=3.561942e-10
x=[-1.71238898044109370, -1.71238897962736900]; ...
    rad=4.068623e-10
x=[1.42915815651462920, 1.42920745034148040]; rad=2.464691e-05
x=[4.57079632633951240, 4.57079633312845870]; rad=3.394473e-09
x=[7.71238895091097290, 7.71238898267473870]; rad=1.588188e-08
```

[55]Detta är lösningen på Problem 3 i [Tu11], s. 105

9 Referenser

9.1 Litteratur

Mo09 - Moore, Ramon E. et al., *Introduction to Interval Analysis*, Society for Industrial and Applied Mathematics, 2009.

Tu11 - Tucker, Warwick, *Validated Numerics - A Short Introduction to Rigorous Computations*, Princeton University Press, 2011.

9.2 Internetbaserade källor

- Gourdon, Xavier and Sebah, Pascal, *Numbers, constants and computation*, http://numbers.computation.free.fr/Constants/Algorithms/newton.html, senast kontrollerad 18-06-2014

9.3 Artiklar och rapporter

- Hansen, E.R., *Publications Related to Early Interval Work of R.E. Moore*, publicerad 13-08-2001 på http:
interval.louisiana.edu, senast kontrollerad 16-06-2014

- Ratz, Dietmar, *Inclusion Isotone Extended Interval Arithmetic - A Toolbox Update*, FCINE Report No. 5/1996, Universität Karlsruhe (TH), Institut fur Angewandte Mathematik, 1996

- TT, Dagens Industri, *Kina har världens snabbaste superdator*, publicerad 17-06-2013, http://www.di.se/artiklar/2013/6/17/kina-har-snabbaste-superdatorn/, senast kontrollerad 18-06-2014