TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKATEADUSKOND
Arvutiteaduse instituut
Informaatika eriala

Karl Kruuse

Ülevaade intervallaritmeetika meetodist

Bakalaureusetöö (6 EAP)

Juhendaja: Professor Eero Vainikko

TARTU 2014

Ülevaade intervallaritmeetika meetodist

Lühikokkuvõte:

Antud töö kirjeldab, mis on intervallaritmeetika ning miks seda vaja on. Töö praktilises osas jälgime LU-faktoriseerimise algoritmi käitumist intervallaritmeetika abil. Teiseks tutvustame ühte intervallaritmeetika laiendust nimega afiinne aritmeetika, mis likvideerib mõningad intervallaritmeetika puudused.

Võtmesõnad:

Intervallaritmeetika, LU-faktoriseerimine, ujukomaarvud, afiinne aritmeetika

Overview of interval arithmetics

Abstract:

This thesis will give an overview of interval aritmetics and why it is important. The thesis practical part observes performance of LU decomopsition algorithm through interval aritmetics. Last section gives an overview of affine arithmetics which is extension of interval arithmetics and is often able to give more accurate results.

Keywords:

interval arithmetics, LU decomposition, floating-point numbers, affine arithmetics

Sisukord

1 Sissejuhatus **4**

2 Ujukomaarvud **5**

3 Mis on intervallaritmeetika? **7**
3.1 Esitus ja omadused 7
3.2 Intervall-laiend 8
3.3 Tehted 8
3.4 Algebralised omadused, mis on sarnased reaalarvude omadustele 9
3.5 Algebralised omadused, mis erinevad reaalarvude omadustest 10
3.6 Ümardamine intervallaritmeetika puhul 11

4 Näide intervallaritmeetika kasutamise kohta: LU-faktoriseerimine Sage arvutuskeskkonnas **12**
4.1 Sage arvutuskeskkond 12
4.2 LU-faktoriseerimine 13
4.3 Sage arvutuskeskkonnas realiseeritud algoritm 13
4.4 Sage'i katsed 14

5 Intervallaritmeetika laiendusi **19**
5.1 Afiinne aritmeetika (*affine arithmetic*) 20
5.1.1 Afiinsed tehted 20
5.1.2 Mitte-afiinsed tehted 21
5.1.3 Afiinse aritmeetika võrdlus intervallaritmeetikaga ja implementatsioonid 21

6 Kokkuvõte **22**

Kirjandus **23**

7 Lisad **25**

1 Sissejuhatus

Tänapäeva arvutites on ujukomaarvud väga levinud. Neid kasutatakse reaalarvude esitamiseks ning on seega väga olulised looduses leiduvate suuruste kujutamisel. Kuna arvutite jõudlus ja kasutatav info hulk on piiratud, siis on oluline valida ujukomaarvule adekvaatne mantissi pikkus. See peaks olema piisavalt pikk, et arvud, mida me kasutame, oleksid täpsed, kuid mitte liiga pikk, et kulutada ressurse liigse täpsuse tagaajamisele. Kuna täpsus võib "kaduma minna" ka arvutuste käigus, siis oleks hea hinnata iga algoritmi jaoks eraldi, kui täpsete arvudega me peaksime arvutama, et tulemus oleks piisavalt informatiivne. Üks moodus seda hinnata on kasutada intervallaritmeetikat.

Tegelikult kasutas juba Archimedes intervallaritmeetikat π arvutamiseks, kus ta hindas ringjoone pikkust välise ja sisemise hulknurga abil. Suurema täpsuse saamiseks suurendas ta hulknurkade nurkade arvu [7]. Kaasaegne intervallaritmeetika töötati välja R. E. Moore poolt peamiselt 1960-ndatel [7], kui see teema muutus aktuaalseks seoses arvutite levikuga.

Käesolevas töös antakse ülevaade intervallaritmeetikast ja uuritakse selle käitumist võrrandsüsteemi lahendamise algoritmi näitel erinevate sisendandmete puhul. Kuna intervallaritmeetika esitatakse ujukomaarvudena ja intervallaritmeetika hindab ujukomaarve, siis teine peatükk räägibki ujukomaarvudest. Kolmas peatükk käsitleb täpsemalt, mis on intervallaritmeetika, kuidas teostada lihtsamaid tehteid ja mainib ära mõned selle tähtsamad omadused. Neljas peatükk kirjeldab üht praktilist eksperimenti intervallaritmeetikaga arvutuskeskkonnas Sage (http://sage.math.ut.ee/). Sage on avatud lähtekoodiga matemaatika tarkvara kogum. See on loodud Pythoni baasil ning selles on sisse ehitatud parameetrina etteantava ujukomaarvude pikkusega intervallaritmeetika moodul. Eksperimendis uurime intervallaritmeetika käitumist maatriksi LU-faktoriseerimise käigus. Me uurime mantissi vajalikku pikkust sõltuvalt oodatava tulemusintervalli laiusest, tulemusintervalli laiuse sõltuvust maatriksi lähenemisest singulaarsele ning uurime tõeliste väärtuste asetsemist saadud intervallides. Kuna tihti kipub tavaintervallaritmeetika hinnang olema laiem arvutuste käigus realiseeruvast, siis on intervallaritmeetika täienduseks töötatud välja täpsemaid meetodeid vea hindamiseks. Viies peatükk räägibki kokkuvõtvalt mõnest intervallaritmeetika laiendusest, andes täpsema ülevaate ühest eist - afiinsest aritmeetikast.

2 Ujukomaarvud

Kuna intervallaritmeetika on loodud täiendamaks ujukomaarvude teatud "puudusi" ja selle esitus on realiseeritud sarnaselt ujukomaarvudele, siis antud peatükis anname ülevaate ujukomaarvudest. Algul kirjeldame, mis on ujukomaarvud, järgnevalt tutvustame levinuimat IEEE 754 standardit ning lõpuks mainime, millised väljakutsed on seotud ujukomaarvudega. Selle peatüki info pärineb allikast [6].

Reaalarvudega arvutamiseks kasutakse tänapäevastes arvutites ujukomaarve. Need koosnevad märgist, oluliseimatest tüvenumbritest ehk mantissist ja astendajast ehk eksponendist, mis näitab, kus koma asetseb. Näiteks π esitamiseks võib kasutada ujukomaarvu $1.10010010000111111011011 \times 2^1$ [10]. Siin on meie arvubaasiks $b = 2$, tüvenumbreid on 24, astendaja $e = 1$ ja arv ise on positiivne ($s = 0$). Mantissi osas on arvu olulisemad tüvenumbrid kujul $d_0.d_1d_2d_3...d_n$, kus $d_0, d_1, d_2, d_3, ..., d_n$ on arvusüsteemi numbrid. Eksponent ehk astendaja näitab mitu kohta ja kummale poole koma liigub. Üldiselt saab ujukomaarvu esituse kokku võtta nii: kui s on märk, m on mantiss, e on astendaja ja b on arvusüsteemi baas, siis ujukomaarv $x = (-1)^s \times m \times b^e$. Ujukomaarvud ongi saanud nime selle järgi, et koma võib oma kohta muuta.

Levinuim ujukomaarvu standard on IEEE 754. See lubab esitada arve ainult kahe baasil. Arvu täpsuseks saab valida kas ühekordse täpsuse (*single precision*), mis kasutab 32 bitti või topelttäpsuse (*double precision*), mis kasutab 64 bitti. Ühekordse täpsuse korral kasutatakse 1 biti märgi jaoks, 8 bitti astendajale ja 23 bitti mantissile. Topelt täpsuse korral on need arvud vastavalt 1, 11, 52. Viimane vorming on kasutusel näiteks Javas tüübina *double* [9].

Ujukomaarvud esitavad meile reaalarvudest kõige olulisema osa, kuid ka see võib arvutuste järel ebatäpseks muutuda. Näiteks kahe väga lähedase arvu lahutamisel oluliste tüvenumbrite arv väheneb tunduvalt. See efekt on tuntud kui tühistamine(i.k. *cancellation*), kus lahutamisel kahe arvu suurema järgu bitid üksteist tühistavad, madalamad bitid nihutatakse kõrgemale ja kuna me ei tea, mis väärtused peaksid saama madalamad bitid, siis täidetakse need lihtsalt nullidega. Sellega aga kaotame arvu täpsuses. Teine veakoht arvutitega arvutades on baasivahetus. Näiteks kümnendarv 0.1 oleks kahendsüsteemis perioodiline $0.0001100110011001100\ldots$. Kui see aga esitada lõpliku kahendmurruna,

siis oleme kaotanud natuke arvu täpsusest.

Suurema hulga arvutuste korral võib see viga kasvada liiga suureks, et arvul enam mingit mõtet oleks. Kuna ujukomaarvu juures puudub info tema täpsusele, siis siin tuleb appi intervallaritmeetika, mis aitab hinnata erinevate arvutuste vea piire.

3 Mis on intervallaritmeetika?

Käesolev peatükk kirjeldab intervallaritmeetikat matemaatilisest seisukohast. See peatükk on koostatud, kasutades allikat [7]. Saame teada, kuidas intervalle esitada, teostada peamisi arvutusi ja tutvume mõne peamise omadusega.

Ujukomaarvudega ei saa me alati esitada soovitud väärtust x, vaid oleme sunnitud kasutama selle lähendit $\dot{x}$. Kui me kasutame arvutustes lähendit $\dot{x}$, siis ei suuda me pärast hinnata, kui kaugele jääb meie saadud tulemus tegelikust soovitud väärtusega x arvutatud tulemusest. Selle vea paremaks hindamiseks kasutatakse intervallaritmeetikat. Intervallaritmeetika ei aita meil täpsemini arvutada, aga ta aitab meil paremini hinnata tulemuse täpsust. Me saame soovitud väärtusele x leida ülemise ja alumise tõkke $\underline{x} \leq x \leq \overline{x}$. Nüüd me teame, et meie soovitud väärtus x on intervallis $X = [\underline{x}, \overline{x}]$. Kui me teostame arvutusi intervalliga X, siis saadud intervall Y peab sisaldama kõiki väärtusi, mis saab X elementidele neid arvutusi rakendades. Intervallaritmeetikat kasutataksegi peamiselt veahinnanguks ja vahemike analüüsimiseks.

3.1 Esitus ja omadused

Intervall on kinnine lõik reaalarvude teljel $X = [a, b]$. Intervalli X alumist ja ülemist tõket tähistatakse tavaliselt $\underline{X}, \overline{X}$. Siit saame

$$X = \left[\underline{X}, \overline{X}\right].$$

Reaalarvud võib lugeda intervallideks, mille ülemine ja alumine tõke langevad kokku. Neid võiks nimetada kidunud intervallideks (degenerated interval). Siin saame ka kirjutada näiteks:

$$7 = [7, 7].$$

Paar olulisemat intervalli näitajat on ka keskpunkt ja laius. Intervalli $X = \left[\underline{X}, \overline{X}\right]$ keskpunkt

$$m(X) = \frac{\underline{X} + \overline{X}}{2}$$

ja laius

$$w\left(X\right)=\overline{X}-\underline{X}.$$

Vahel esitatakse intervallid ka keskpunkti ja laiuse abil:

$$X=m(X)+\frac{w\left(X\right)}{2}\left[-1,1\right].$$

Kahte intervalli $X=\left[\underline{X},\overline{X}\right]$ ja $Y=\left[\underline{Y},\overline{Y}\right]$loetakse võrdseteks $X=Y$, kui $\underline{X}=\underline{Y}$ ja $\overline{X}=\overline{Y}$.

Intervall $X=\left[\underline{X},\overline{X}\right]$ on väiksem kui intervall $Y=\left[\underline{Y},\overline{Y}\right]$ juhul, kui $\overline{X}<\underline{Y}$.

3.2 Intervall-laiend

Rakenduslikult ei ole alati reaalarvude hulgal määratud funktsiooni kujutus intervalist lihtsasti leitav. Siis kasutatakse hoopis intervall laiendit, mis on enamasti lihtsam kasutada, kuid mis võib olla laiem, kui tegelik kõigi võimalike väärtuste hulk. Funktsiooni $F:[\mathbb{R}]\to[\mathbb{R}]$ nimetatakse funktsioon $f:\mathbb{R}\to\mathbb{R}$ intervall laiendiks kui mingi intervalli X korral

$$F\left(X\right)\supseteq\left\{f\left(x\right):x\in X\right\}.$$

Samuti kehtib see mitmemuutuja funktsiooni korral. N-muutuja funktsiooni $F:[\mathbb{R}]^n\to[\mathbb{R}]$ nimetatakse funktsiooni $f:\mathbb{R}^n\to\mathbb{R}$ intervall-laiendiks, kui

$$F\left(X_1,X_2,\ldots,X_n\right)\supseteq\left\{f\left(x_1,x_2,\ldots,x_n\right):x_1\in X_1,x_2\in X_2,\ldots,x_n\in X_n\right\}.$$

Funktsioon $F\left(X\right)=(-\infty,\infty)$ sobib küll kõigi funktsioonide intervall-laiendiks, kuid ei ole väga informatiivne. Enamasti leidub siiski funktsioonidel intervall-laiend, mis on sama täpne intervalli kujutisega reaalarvude hulgal määratud funktsiooniga. Järgmises punktis toome elementaartehete intervall laiendid.

3.3 Tehted

Kui mingi funktsioon $f(x)=y$ on määratud realarvude teljel $\mathbb{R}$, siis iga intervalli korral $X=\left[\underline{X},\overline{X}\right]\subseteq\mathbb{R}$ saame öelda,

$$f(X)=\left\{f(x):x\in X\right\}.$$

Sarnaselt on defineeritud tehted intervallide $X = \left[\underline{X}, \overline{X}\right]$ ja $Y = \left[\underline{Y}, \overline{Y}\right]$ vahel

$$X \odot Y = \{x \odot y : x \in X, y \in Y\}.$$

Funktsiooni $f(X)$ leidmiseks ei pea me alati leidma kõiki $f(x) : x \in X$ väärtusi. Kui funktsioon on pidev, siis piisab sellest, kui me leiame funktsiooni f väärtused intervalli otspunktides ja ekstreemumites, kui neid leidub. Nendest väärtustest vähim on meie intervalli alguspunkt ja suurim intervalli lõpp-punkt.

Liitmist, lahutamist, korrutamist ja jagamist kahe intervalli $X = \left[\underline{X}, \overline{X}\right]$ ja $Y = \left[\underline{Y}, \overline{Y}\right]$ vahel on küllalt kerge sooritada ka ülemise ja alumise raja abil kus,

$$X + Y = [\underline{X} + \underline{Y}, \overline{X} + \overline{Y}]$$

$$X - Y = [\underline{X} - \overline{Y}, \overline{X} - \underline{Y}]$$

$$X \times Y = \left[min\left(\underline{X} \times \underline{Y}, \underline{X} \times \overline{Y}, \overline{X} \times \underline{Y}, \overline{X} \times \overline{Y}\right), max\left(\underline{X} \times \underline{Y}, \underline{X} \times \overline{Y}, \overline{X} \times \underline{Y}, \overline{X} \times \overline{Y}\right)\right]$$

$$X/Y = \left[\underline{X}, \overline{X}\right] \times \left[1/\overline{Y}, 1/\underline{Y}\right], \ 0 \notin Y.$$

Jagamine on lubatud vaid juhul, kui jagaja-intervall ei sisalda arvu 0.

Intervalli $X = \left[\underline{X}, \overline{X}\right]$ absoluutväärtus $\|X\| = \max\left(|\ \underline{X}\ |, |\ \overline{X}\ |\right)$.

3.4 Algebralised omadused, mis on sarnased reaalarvude omadustele

Intervallaritmeetika on liitmise ja korrutamise suhtes kommutatiivne ja assotsiatiivne:

$$X + Y = Y + X,$$

$$XY = YX,$$

$$X + (Y + Z) = (X + Y) + Z,$$

$$X\left(YZ\right)=\left(XY\right)Z.$$

Intervallide hulgas leidub null- ja ühikelement. Nullelement on intervall, mis sialdab vaid arvu 0, $0=\left[\underline{0},\overline{0}\right]$. Ühikelement on intervall, mis sisaldab vaid arvu 1, $1=\left[\underline{1},\overline{1}\right]$. Interval, 0- ja ühikelemendi olemasolu

$$0+X=X+0=\left[\underline{X},\overline{X}\right]+\left[\underline{0},\overline{0}\right]=\left[\underline{X}+\underline{0},\overline{X}+\overline{0}\right]$$

$$1\times X=X\times 1=\left[\underline{X},\overline{X}\right]\times\left[\underline{1},\overline{1}\right]=\left[\underline{X},\overline{X}\right].$$

3.5 Algebralised omadused, mis erinevad reaalarvude omadustest

Vastandelement nii liitmise kui korrutamise suhtes leidub ainult kidunud intervallidel ehk intervallidel, mis sisaldavad vaid üht arvu. Üldjuhul:

$$X+\left(-X\right)=\left[\underline{X},\overline{X}\right]+\left[-\overline{X},-\underline{X}\right]=\left[\underline{X}-\overline{X},\overline{X}-\underline{X}\right]=w\left(X\right)\left[-1,1\right],$$

$$X/X=\begin{cases}\left[\underline{X}/\overline{X},\overline{X}/\underline{X}\right],\ \text{kui}\,0<\underline{X}\\ \left[\overline{X}/\underline{X},\underline{X}/\overline{X}\right],\ \text{kui}\,\overline{X}<0\end{cases}.$$

Intervallide puhul distributiivsus-seadus $X(Y+Z)=XY+XZ$, ei kehti. Kui võtta näiteks $X=\left[1,2\right],\ Y=\left[1,1\right],\ Z=\left[-1,-1\right]$, siis

$$X(Y+Z)=X\left(\left[1,1\right]+\left[-1,-1\right]\right)=\left[1,2\right]\times 0=\left[0,0\right]$$

$$XY+XZ=\left[1,2\right]\times\left[1,1\right]+\left[1,2\right]\times\left[-1,-1\right]=$$

$$=\left[\min\left(1,2\right),\max\left(1,2\right)\right]+\left[\min\left(-1,-2\right),\max\left(-1,-2\right)\right]=$$

$$=\left[1,2\right]+\left[-2,-1\right]=\left[-1,1\right].$$

Intervallide puhul kehtib alamdistributiivsuse seadus:

$$X(Y+Z)\subseteq XY+XZ.$$

Siiski kehtib ka distributiivsusseadus teatud juhtudel. Näiteks, kui $X=[x,x]$ on kidu-

nud intervall ehk reaalarv:

$$x(Y + Z) = xY + xZ.$$

Samuti, kui $YZ > 0$, ehk, Y ja Z on samamärgilised, siis

$$X(Y + Z) = XY + XZ, \; kui \, YZ > 0.$$

3.6 Ümardamine intervallaritmeetika puhul

Intervallaritmeetikat kasutatakse peamiselt olukordades, kus täpsed arvutused ei ole võimalikud. Kui me ei saa intervalli üht otspunkti esitada täpselt (näiteks arvusüsteemi ja mälu piiratuse tõttu) ja oleme sunnitud ümardama, siis peaksime ümardama intervalli laiemaks, et tegelik intervall sisalduks selles ümardatud intervallis. Selle tagamiseks peab alati intervalli alumise tõkke ümardama allapoole ja ülemise tõkke ülespoole.

Kui me ei saa intervalli X otspunkte täpselt esitada ja oleme sunnitud ümardama $R(X) \approx X$, siis peame seda tegema nii, et $X \subseteq R(X)$.

Eelnevalt tõime lühiülevaate intervallaritmeetika omadustest, järgmises peatükis toome ühe konkreetse arvutusliku näite.

4 Näide intervallaritmeetika kasutamise kohta: LU-faktoriseerimine Sage arvutuskeskkonnas

Käesolevas peatükis uurime Sage arvutusserveri sisseehitatud intervallaritmeetika mooduli (*RealIntervalField*) käitumist, kasutades LU-faktoriseerimise algoritmi. Peamine küsimus on, kui laia mantissi osa peab intervallaritmeetikale määrama, et meie algoritmi tulemus jääks etteantud laiusega vahemikku. Samas saame jälgida algoritmi käitumist intervallaritmeetika abil, kui arvutustes kasutatav maatriks läheneb singulaarsele maatriksile.

Teeme arvutuskeskkonnas Sage mõned praktilised katsed intervallaritmeetikaga. Algul tutvustame, mis on Sage ning kuidas on seal realiseeritud intervallaritmeetika. Edasi anname ülevaate arvutusteks kasutatud LU-faktoriseerimise algoritmist. Seejärel tutvustame katseid ja saadud tulemusi.

4.1 Sage arvutuskeskkond

Selle alampeatüki sisu tugineb allikal [12]. Sage on Pythoni baasil kirjutatud avatud lähtekoodiga matemaatika tarkvara. See toetub mitmele avatud lähtekoodiga pakettile näiteks NumPy, Sympy, SciPy, R, matplotlib, Maxima, GAP, FLINT jne. [4]. Tartu Ülikooli Sage server on kättesaadav ülikooli sisevõrgust aadressil http://sage.math.ut.ee/.

Sage's on reaalarvuliste ujukomaarvude jaoks kaks erinevat klassi. Lihtsam, IEEE 754 standardile vastav *Double Precision Real Numbers* (ka *RealDoubleField*), on sarnane Pythoni *float* tüübi ja näiteks Java *double* tüübiga ning selle mantissi pikkus on 53 bitti. Kümnendarvudesse teisendatult on see umbes 15-17 tüvenumbrit, kuid kui liita *RealDoubleField* klassis arve $1 + 10^{-12}$, siis tulemuseks on 1, mis tähendab, et antud klassil hakkab ilmnema olulisi täpsuse vigu juba 12 tüvenumbri juures. Teatud liiki teadusarvutustes, mida Sage soovib toetada, võib sellest aga puudu jääda. Selliste arvutuste jaoks on sobivam teine, suvalise täpsusega arvuklass *Field of Arbitrary Precision Real Numbers* (ka *RealField*). Nagu nimigi ütleb, saab selle klassi isenditel mantissi pikkuse ise määrata. Sellele sarnane on ka intervallaritmeetika klass *Field of Arbitrary Precision*

Real Intervals (ka *RealIntervalField*), milles iga arv salvestatakse kahe määratud mantissi pikkusega ujukomaarvudena, millest esimene on arvu alamtõke ning teine ülemtõke. Meie katsetused saavadki olema teostatud kasutades *RealIntervalField* klassi.

4.2 LU-faktoriseerimine

LU-faktoriseerimine ehk Gaussi elimineerimise meetod [8] on võrrandisüsteemide lahendamise meetod, kus võrrandisüsteemi maatriks A teisendatakse kaheks lihtsama struktuuriga maatriksi korrutiseks $A = LU$, kus L on alamkolmnurkmaatriks, mille peadiogonaali peal olevad elemendid on nullid ja peadiagonaalil on ühed. U on ülemkolmnurkmaatriks mille peadiagonaali all olevad elemendid on nullid. Vahel, kui faktoriseerimise käigus maatriks A diagonaalil ilmneb nulle, lisatakse sinna ka kolmas, permutasioonimaatriks P, millega vahetatakse vajadusel mõned read või veerud, et L ja U oleksid lihtsamine leitavad. Sellisel juhul teiseneks võrrandisüsteem $A = PLU$. Permutatsioonimaatriks P muudab arvutused numbriliselt stabiilsemaks. Käesolevas töös me lihtsuse mõttes permutatsioone siiski ei tee. Seega võrrandisüsteemist $Ax = b$ saame $LUx = b$ ning siit saab leida $x = U^{-1}L^{-1}b$.

Kui A peadiagonaalil on kõik elemendid nullist erinevad ($a_{ii} \neq 0$), siis saab L ja U leida järgneva algoritmi abil:

1. Olgu A n-ndat järku ruutmaatriks, siis alustades esimesest reast kuni viimaseni $i \in \{1, 2, \ldots, n\}$:

    1.1. Kõigist järgnevatest ridades lahutame $\frac{a_{ji}}{a_{ii}}$ kordse i-nda rea: $a_j = a_j - a_i \times \frac{a_{ji}}{a_{ii}},\ j \in \{i+1, \ldots, n\}$

    1.2. Maatriksisse L lisame elemendi $l_{ji} = \frac{a_{ji}}{a_{ii}}$

2. nimetame muudetud maatriksi A ümber U-ks.

4.3 Sage arvutuskeskkonnas realiseeritud algoritm

Käesolev peatükk kirjeldab Sage's realiseeritud LU-faktoriseerimise algoritmi koos võr-

randisüsteemi maatriksi loomise ja võrrandisüsteemi lahendamisega.

1. Genereerime $n \times n$ maatriksi A, mille elementideks on kidunud intervallarvud vahemikust [0..1].

2. Peadiagonaalile kirjutame rea ülejäänud elementide negatiivse summa + väga väike ϵ, et meie loodud maatriks oleks singulaarse maatriksi "sarnane":

$$A_{kk} = -\left(\sum A_{ki}\right) + \epsilon, \text{ kus } i \in \{1, 2, \ldots, n\}, i \neq k.$$

3. Genereerime $n \times 1$ maatriksi X^{teada}, mille elementideks on kidunud intervallarvud vahemikust [0..1].

4. Arvutame maatriksi $b = A \times X^{teada}$, et hakata lahendama võrrandsüsteemi $AX = b$.

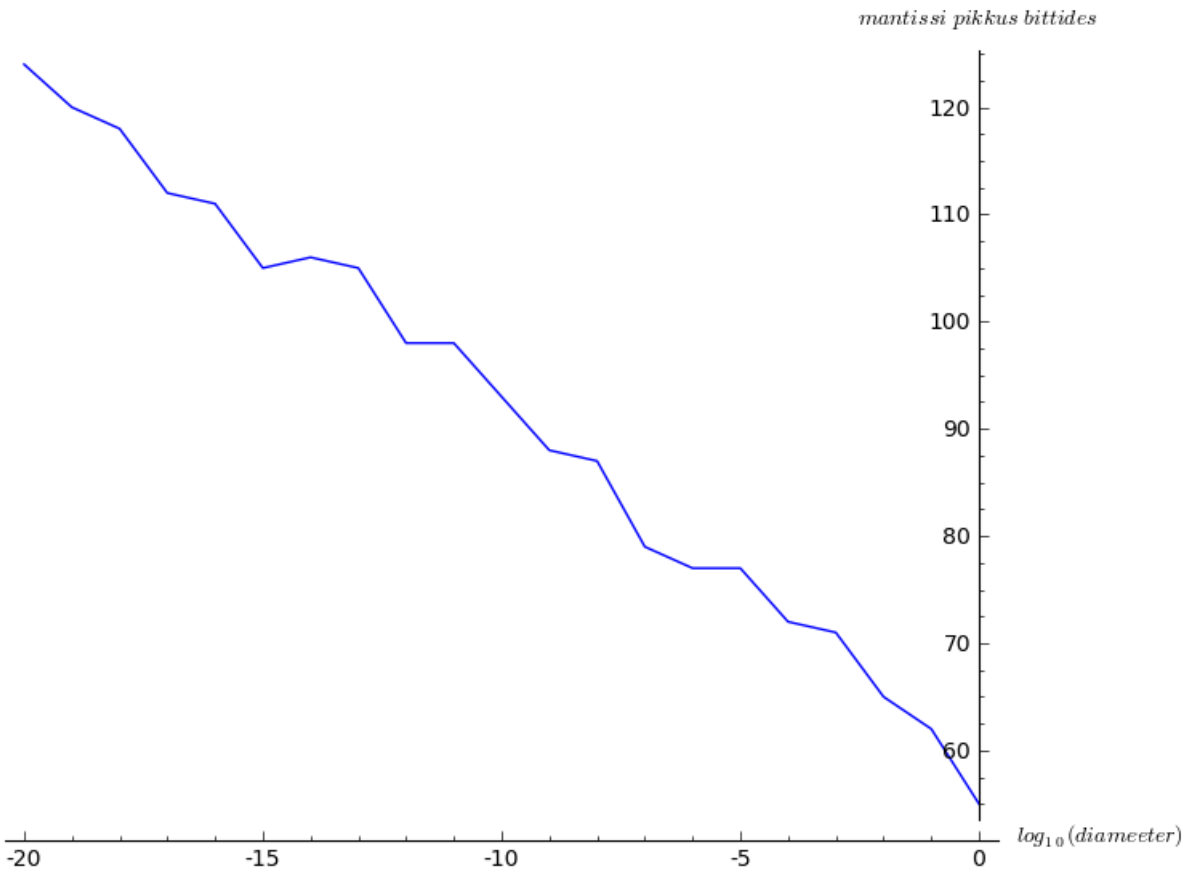
5. Nüüd püüame lahendada eelmises punktis mainitud võrrandit kasutades LU faktoriseerimist, $X = A^{-1}b = U^{-1}L^{-1}b$.

6. Kui oleme leidnud maatriksi X, leiame sealt suurima intervalli laiuse ning tagastame selle.

4.4 Sage'i katsed

Kuigi huvitav oleks teada LU-faktoriseerimise käitumist maatriksitega, mille järk on 1000 ringis, siis TÜ Sage serveri võimekuse tõttu peame tegema katseid tagasihoidlikumate sisendandmetega.

Meid huvitab, kui pika mantissi osa me peaksime *RealIntervalField* klassil valima, et meie 8×8 maatriksi LU-faktoriseerimise tulemuse intervalli laius (saadud maatriksi elementide suurim laius) oleks väiksem soovitud suurusest. Kuna me tahame näha tüübi *RealIntervalField* eeliseid tüübi *RealDoubleField* ees, siis määrame $\epsilon = 10^{-12}$. Meenutuseks punktist 4.1, kui tegeleda arvudega vahemiskus $[0 \ldots 1]$, siis see on *RealDoubleField* klassi jaoks piir, kus võib hakata juhtuma, et järgnevad tüvenumbrid ei ole täpsed. Vajaliku mantissiosa pikkuse leiame katsetamise teel. Teeme 21 katset, alustades laiusest 1 ning iga järgnev katse on 10 korda kitsama intervalli peal. Seega vaadeldavad laiused jäävad hulka $w = 10^x,\ x \in \{-20, 0\}$. Igal katsel rakendame punktis 4.3 toodud algoritmi, alustades mantissi pikkusega 15 ning seda ühe võrra suurendades, kuni meie LU-faktoriseerimisel saadud laius on väiksem ette antud laiusest. Tulemused on toodud

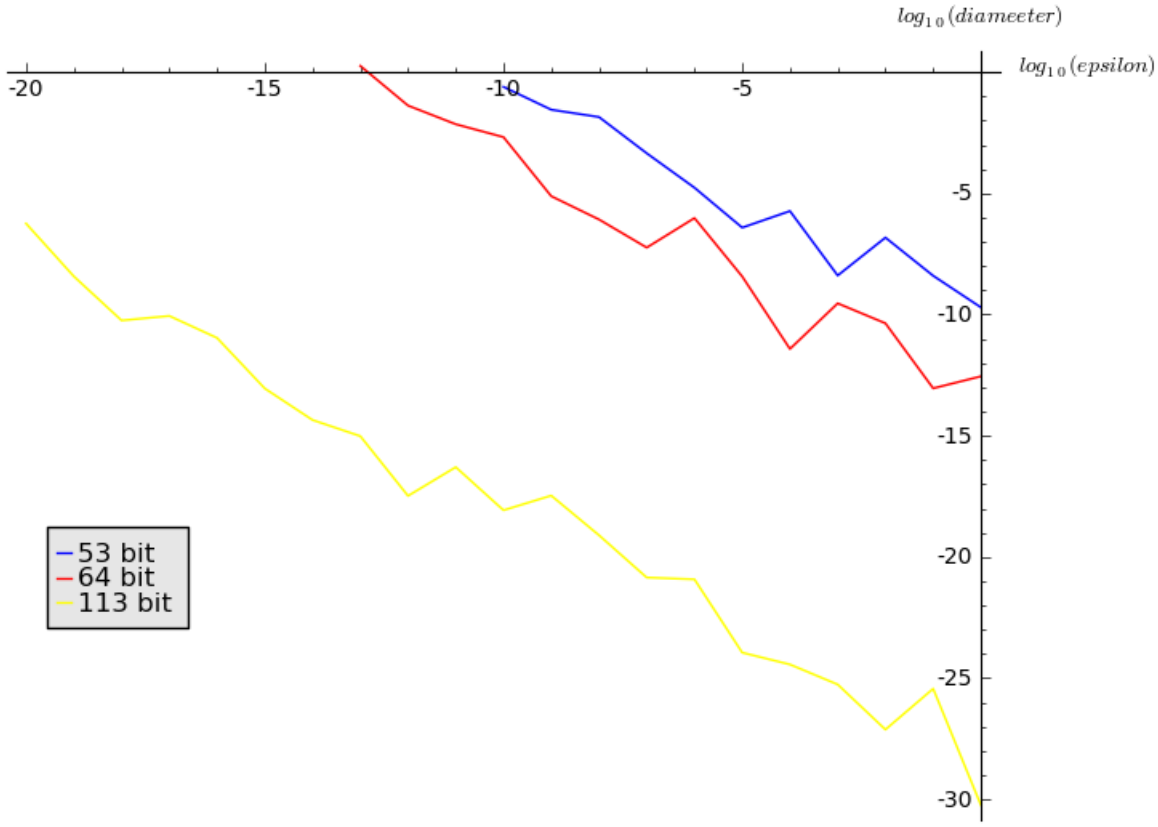


Joonis 4.1: Mantissi vajalik pikkus tulemuse ette antud diameetrisse mahutamiseks

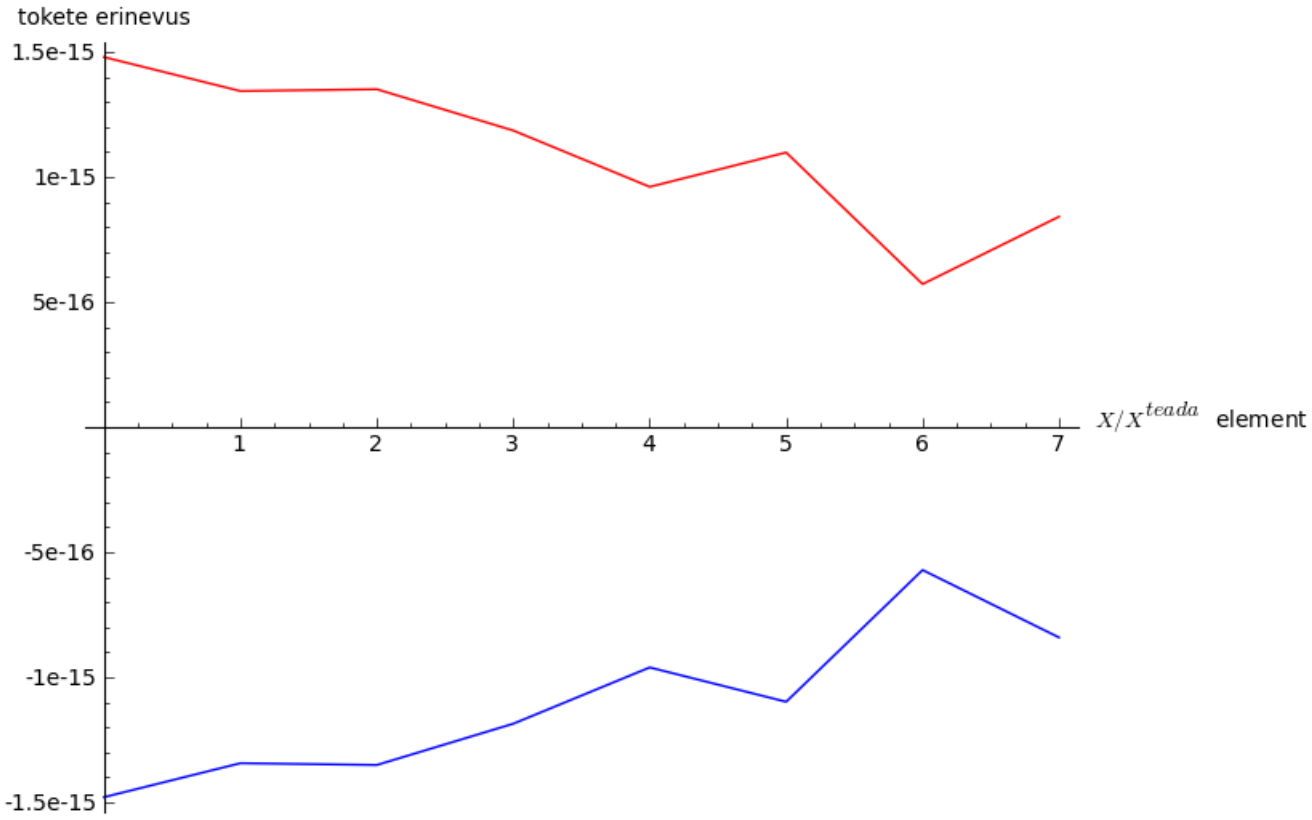
Joonisel 4.1. Siit näeme, et mantissi vajalik pikkus on lineaarses seoses kümnendkoma-kohtade arvuga. Antud katses läks ühe kümnendkoha lisamiseks tulemuse täpsusele vaja keskmiselt $\frac{123-55}{20} = 3.4$ bitti mantissi osas.

Teiseks soovime näha kindla mantissi pikkuse korral tulemusintervalli diameetri käitumist maatriksi lähenemisel singulaarsele, ehk punktis 4.3 mainitud ϵ lähenemisel 0-le. Joonisel 4.2 sinine joon vastab mantissi pikkusele 53, punane 64, ja kollane 113 bitti, millest 53 ja 113 on IEEE-754 standardi järgi vastavalt 64 ja 128 bitise formaadi mantissi osad [1]. Sel katsel 8×8 maatriksiga laienes LU-faktoriseerimise tulemusintervall kümme korda, kui epsilon vähenes kümme korda.

Kolmandaks tahame teada, kuidas paikneb meie algselt genereeritud X^{teada} LU-faktoriseerimise kaudu arvutatud intervalli X suhtes. Selleks rakendame punktis 4.3 toodud algoritmi 8×8 maatriksile ning vaatame saadud 8×1 maatriksi iga elemendi kohta, kui palju on ülemise



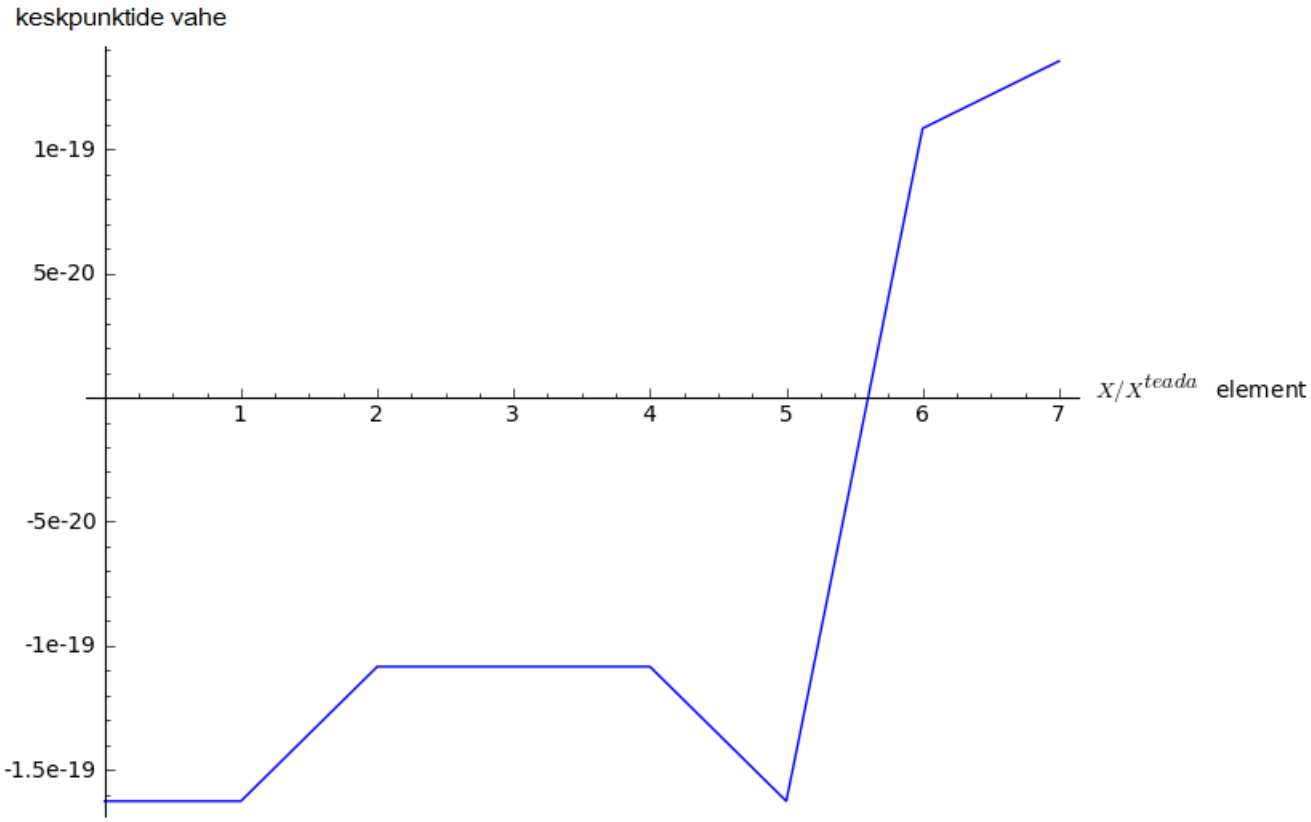
Joonis 4.2: Tulemus intervalli diameeteri sõltuvus *epsilonist*, ehk rea elementide summa erinevusest 0-st



Joonis 4.3: X tõkete võrdlus X^{teada} tõketega

ja alumise tõkke vahe algse, genereeritud maatriksi vastavalt ülemise või alumise tõkkega. Et intervalli laiuse muutus oleks paremini näha, peaksime valima sisendparameetrid nii, et tulemusintervall oleks väiksema diameetriga. Nagu eelmistest katsetest teame, peaks selleks võtma küllalt suure mantissiosa ja võimalikult suure epsiloni. Joonisel 4.3 on tulemus 64 bitise mantissiosa ja $\epsilon = 10^{-1}$ puhul tehtud katsest. Punane joon näitab, kui palju on X elemendi ülemine tõke suurem kui X^{teada} vastava elemendi ülemine tõke $U = \overline{X_i} - \overline{X_i^{teada}}$. Sinine joon näitab, kui palju on X elemendi alumine tõke suurem kui X^{teada} vastava elemendi alumine tõke $L = \underline{X_i} - \underline{X_i^{teada}}$. Nagu jooniselt näha, paiknevad maatriksi X^{teada} elemendid üsna täpselt maatriksi X elementide keskel.

Kuna joonise 4.3 järgi on X arvutuste käigus nii ülalt kui alt poolt kasvanud küllaltki võrdselt, siis oleks hea teada kummas suunas ja kui palju liikus intervalli keskpunkt? Joonisel 4.4 on intervallide X ja X^{teada} keskpunktide vahe. Siit me näeme, et kui me rakendame LU-faktoriseerimist 8×8 maatriksil 64 bitise mantissi pikkusega intervallaritmeetikal ja meie singulaarsuse näitaja on $\epsilon = 10^{-1}$, siis arvutatud intervalli keskpunkt



Joonis 4.4: X keskpunktide kaugus X_teada keskpunktidest

erineb algsest 1.6×10^{-19}. Meenutuseks jooniselt 4.2, 64 bitise mantissi pikkusega ja $\epsilon = 10^{-1}$ korral tuli intervalli laiuseks 10^{-13}.

Kuna viimase katse tulemusintervalli laius (10^{-13}) on 100000 korda suurem, kui intervallide X ja X^{teada} keskpunktide vahe, siis võib öelda, et intervallaritmeetika hindab võimalikke tulemuste hulka üle. Seepärast on püütud intervallaritmeetikat täiustada, et saada arvudele täpsemaid hinnanguid. Järgmises peatükis räägimegi ühest intervallaritmeetika laiendusest.

5 Intervallaritmeetika laiendusi

Peamine põhjus intervallaritmeetika täiendamiseks on olnud selle tulemusintervallide ülehindamine. Mitme parameetriga funktsiooni intervallaritmeetikas arvutatud inetrvall kipub olema laiem, kui kõikvõimalike parameetrite komplektide korral arvutatud väärtuste hulk. See juhtub enamasti siis, kui erinevad parameetrid on tegelikult omavahel seotud. Näiteks $X \times X$ intervalli $X = [-1, 1]$ korral saame tulemuseks

$$X \times X = [-1, 1] \times [-1, 1] = [-1, 1],$$

kuna

$$\underline{X \times X} = min(-1 \times (-1), -1 \times 1, 1 \times (-1), 1 \times 1) = -1$$

ja

$$\overline{X \times X} = max(-1 \times (-1), -1 \times 1, 1 \times (-1), 1 \times 1) = 1.$$

Samas, kui me leiame intervalli $X = [-1, 1]$ kujutise funktsiooniga $f(x) = x \times x$ saame kujutuseks $[0, 1]$. See tuleneb sellest, et esimesel juhul me vaatame kõik võimalikke kombinatsioone, kus valitakse hulgast X kaks elementi. Teisel juhul me arvestame, et tegu on sama arvuga ehk et nad on omavahel võrdsõltuvuses.

Teine levinud põhjus intervallide liigseks kasvamiseks on taandamata võrrandid. Reaalarvude aritmeetikas võib valemile vabalt lisada mingi arvu vahe iseendaga $a - a = 0$. Intervallaritmeetikas aga see ei kehti, sest

$$A - A = \left[\underline{A}, \overline{A}\right] - \left[\underline{A}, \overline{A}\right] = \left[\underline{A} - \overline{A}, \overline{A} - \underline{A}\right].$$

Kui reaalarvude aritmeetikas $f(x) = f(x) + 0 = f(x) + a - a$, siis intervallaritmeetikas $f(x) = f(x) + 0 \subseteq f(x) + A - A = f(x) + \left[\underline{A} - \overline{A}, \overline{A} - \underline{A}\right]$. See aga kasvatab liigselt tulemuse intervalli.

Intervallaritmeetika laiendused püüavadki vähendada seotud muutujatega tehtud tehetest tulenevaid ülehinnanguid. Omavahel on üsna sarnased **üldistatud intervallaritmeetika** (***generalized interval arithmetics***) ja **afiinne aritmeetika** (***affine aritmetic***s). Lisaks leidub veel laiendatud intervallaritmeetika (*extended interval arithmetics*)

[7], esimest järku Taylori aritmeetika (*first-order Taylor arithmetic*s), kesktõusu meetod (*center-slope model*) ja ellipsoidarvutused (*ellipsoid calculus*) [11].

5.1 Afiinne aritmeetika (*affine arithmetic*)

Kuna üldistatud intervallaritmeetika ja afiine aritmeetika on väga sarnased, siis kirjutame siin kohal täpsemalt afiinsest aritmeetikast. Siin kirjeldame, mis on afiinne aritmeetika ja anname lühikese ülevaate, kuidas seal arvutada. Järgnev tekst afiinse aritmeetika kohta tugineb allikale [11].

Afiine aritmeetika püüab jälgida ja meelde jätta, milliste muutujate baasil on väärtused leitud. Järgnevates tehetes peaksime teadma nende uute väärtuste omavahelisi seoseid. Afiinses aritmeetikas esitatakse arvud afiinse vormi kaudu $x = x_0+x_1\epsilon_1+x_2\epsilon_2+\cdots+x_n\epsilon_n$, kus $x_0, x_1, \ldots, x_n$ on reaalarvud ja $\epsilon_1, \epsilon_2, \ldots, \epsilon_n$ on mürasümbolid ehk sümboolsed intervallarvud $[-1, 1]$ ja x_0 on intervalli keskpunkt. Afiinsest aritmeetikast intervallaritmeetikasse teisendades saame kasutada afiinse vormi liiget x_0 intervalli keskpunktina ja $2 \times \sum \mid x_i \mid$, $i \in \{1, \ldots, n\}$ intervalli laiusena. Näiteks afiinne vorm $x = 1 + \epsilon_1$ esitab intervalli $[0, 2]$. Kui kahel arvul leidub mingi $i > 0$ korral $x_i > 0$, siis on nad omavahel sõltuvuses. Näiteks arvud $x = 1 + \epsilon_1 = [0, 2]$ ja $y = 4 - 2\epsilon_1 + \epsilon_2 = [1, 7]$ on oma vahel seotud ϵ_1 kaudu.

5.1.1 Afiinsed tehted

Liitmist ja skalaariga korrutamist kutsutakse afiinses aritmeetikas ka afiinseteks teheteks, kuna nende rakendamisel ei lisandu mürasümboleid ehk uusi seoseid. Olgu $x = x_0+x_1\epsilon_1+x_2\epsilon_2+\cdots+x_n\epsilon_n$, $y = y_0 + y_1\epsilon_1 + y_2\epsilon_2 + \cdots + y_n\epsilon_n$ afiinsed vormid, α, β, $\gamma \in \mathbb{R}$, siis

$$z = \alpha x + \beta y + \gamma =$$

$$= \alpha\left(x_0 + x_1\epsilon_1 + x_2\epsilon_2 + \cdots + x_n\epsilon_n\right) + \beta\left(y_0 + y_1\epsilon_1 + y_2\epsilon_2 + \cdots + y_n\epsilon_n\right) + \gamma =$$

$$= \alpha x_0 + \beta y_0 + \gamma + \alpha x_1\epsilon_1 + \beta y_1\epsilon_1 + \alpha x_2\epsilon_2 + \beta y_2\epsilon_2 + \cdots + \alpha x_n\epsilon_n + \beta y_n\epsilon_n.$$

Siit saame, et $z_0 = \alpha x_0 + \beta y_0 + \gamma$ ja $z_i = \alpha x_i\epsilon_i + \beta y_i\epsilon_i$, $i \in \{1, \ldots, n\}$.

Näide: Kui $x = 1 + \epsilon_1 = [0, 2]$ ja $y = 4 - 2\epsilon_1 + \epsilon_2 = [1, 7]$, siis

$$x + y = 1 + \epsilon_1 + 4 - 2\epsilon_1 + \epsilon_2 = 5 - \epsilon_1 + \epsilon_2 = [3, 7]\,.$$

5.1.2 Mitte-afiinsed tehted

Kõikide tehete puhul ei saa me alati tulemuseks täpseid afiinseid vorme. Selliste tehete puhul peab leidma afiinse lähendi, mis jätaks alles senised seosed ning lisaks uue mürasümboli. Saadud lähendi valimine on küllalt vaba, kuid peab jälgima, et tulemusest saadud intervall oleks laiem, kui tehte kujutis lähte-intervallide hulgal. Mitte-afiinse funktsiooni $z = f(x, y)$ puhul tuleb leida polünoomide $x = x_0 + x_1\epsilon_1 + x_2\epsilon_2 + \cdots + x_n\epsilon_n$ ja $y = y_0 + y_1\epsilon_1 + y_2\epsilon_2 + \cdots + y_n\epsilon_n$ kaudu z-le vastav afiinne vorm $z = z_0 + z_1\epsilon_1 + z_2\epsilon_2 + \cdots + z_n\epsilon_n$. Enamasti saame leida $z_i = f_i(x, y)$, $i \in \{0, \ldots, n\}$ ning lisandub uus vaba mürasümbol ϵ_k, ning sellele vastav kordaja z_k. Näiteks $z = x \times y$ puhul sobiks afiinseks lähendiks

$$z_0 = x_0 + y_0$$

$$z_i = x_i \times y_0 + x_0 \times y_i$$

ning lisandub vaba mürasümbol

$$z_{n+1} = \mid (x_1 + x_2 + \cdots + x_n)(y_1 + y_2 + \cdots + y_n) \mid,$$

kokkuvõtvalt

$$x \times y = x_0 + y_0 + (x_1 \times y_0 + x_0 \times y_1)\epsilon_1 + \ldots + (x_n \times y_0 + x_0 \times y_n)\epsilon_n +$$

$$+ \mid (x_1 + x_2 + \cdots + x_n)(y_1 + y_2 + \cdots + y_n) \mid \epsilon_k.$$

Mitte-afiinsete tehete täpsus sõltub suuresti afiinse lähendi valikust.

5.1.3 Afiinse aritmeetika võrdlus intervallaritmeetikaga ja implementatsioonid

Nagu näha, siis afiinne aritmeetika püüab täpsema hinnangu saamiseks meelde jätta erinevate muutujate vahelisi seoseid. See info on eriti kasulik, kui samu muutujaid kasutatakse korduvalt, näiteks rekursiivsetel funktsiooni kasutustel. See muudab ka arvutused palju täpsemaks ja tulemusintervall ei kasva arvutuste käigus liigselt. See kõik tuleb muidugi keerukamate arvutuste arvelt. Afiinse aritmeetika jaoks leiduvad implementatsioonid C++ pakettidena. Näiteks Libaffa[5], Aaflib[2] ja YalAA[3]. Hetkel Sages puudub afiinse aritmeetika tugi.

6 Kokkuvõte

Antud töös tutvusime intervallaritmeetikaga. Alustasime intervallaritmeetika vajadusest rakenduslike arvutuste juures ja implementatsioonist ujukomaarvude kaudu. Peamised puudused rakenduslike arvutuste juures on nende piiratus (lõplikkus) ja baasi vahetus kümnend- ja kahendsüsteemi vahel. Vaatasime lihtsamaid tehteid ja algebralisi omadusi matemaatilisest küljest ning seost intervallaritmeetika ja tavaaritmeetika vahel, mis toiumub peamiselt läbi intervall-laiendi. Töö neljandas osas uurisime intervallaritmeetika käitumist eksperimentaalselt arvutusserveris Sage. Meie katsed toimusid LU-faktoriseerimise baasil. Tulemustena leidsime, et vajaliku mantissi osa pikkus bittides on lineaarses sõltuvuses soovitud tulemusintervalli laiuse kümnend komakohtade arvuga. Samuti nägime, et tulemusintervalli laiuse kümnendkohtade arv oli lineaarses seoses maatriksi rea elementide summa kümnendkohtadega. See viitab sellele, et mida sarnasem on maatriks singulaarsele maatriksile, seda laiemaks muutub tulemusintervall. Viimasena saime teada, et 8×8 maatriksi LU-faktoriseerimisel saadud intervallide laiused olid umbes 100000 korda suuremad, kui nende intervallide keskpunktide vahe teadaolevate, tegelike väärtustega. Kuna see näitab, et intervallaritmeetika hindab tulemusi ebatäpsemaks, kui nad muidu on, siis uurisime ka intervallaritmeetika laiendusi, mis aitaksid saada täpsemaid tulemusi. Töö mahu piiratuse tõttu jõudime põhjalikumalt tutvuda neist vaid ühega. Viimases sisulises osas rääkisimegi afiinsest aritmeetikast, afiinsest vormist ning kuidas sellega teostada afiinseid ning mitte-afiinseid tehteid. Kuna intervallaritmeetika on pigem esimene samm matemaatika ja rakenduslike arvutuste vahel, siis täpsemate tulemuste saamiseks võiks uurida edasi pigem afiinset aritmeetikat ning selle rakendusvõimaluste loomiseks näiteks Sage arvutuskeskkonnas.

Kirjandus

[1] *IEEE Standard for Floating-Point Arithmetic.* http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4610935. Version: August 2008. – Viimati külastatud 2014-07-30

[2] *aaflib - An Affine Arithmetic C++ Library.* http://aaflib.sourceforge.net/. Version: Oktober 2010. – Viimati külastatud 2014-07-16

[3] *YalAA: YalAA - Yet Another Library for Affine Arithmetic.* http://www.scg.inf.uni-due.de/fileadmin/Projekte/YalAA/index.html. Version: September 2012. – Viimati külastatud 2014-07-16

[4] *Sage: Open Source Mathematics Software.* http://www.sagemath.org/. Version: 2014. – Viimati külastatud 2014-05-27

[5] Gay, Olivier ; Coeurjolly, David ; Hurst, Nathan J.: *Libaffa - C++ Affine Arithmetic Library for GNU/Linux.* http://www.nongnu.org/libaffa/. Version: Dezember 2004. – Viimati külastatud 2014-07-16

[6] Goldberg, David: *What Every Computer Scientist Should Know About Floating-Point Arithmetic.* http://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html. Version: März 1991. – Viimati külastatud 2014-05-26

[7] Kearfott, R. B. ; Moore, Ramon E. ; Cloud, Michael J.: *Introduction to interval analysis.* http://www.sbras.ru/interval/Library/InteBooks/IntroIntervAn.pdf. Version: 2009. – Viimati külastatud 2014-05-26

[8] Miidla, Peep: *Lineaarsete võrrandisüsteemide lahendamine.* http://math.ut.ee/~peepm/IMmodels/failid/Lineaarsete_vorrandis%C3%BCsteemide_lahendamine.ppt. – Viimati külastatud 2014-07-17

[9] Oracle: *Primitive Data Types (The Java™ Tutorials > Learning the Java Language > Language Basics).* http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html. Version: 2014. – Viimati külastatud 2014-05-26

[10] Regan, Rick: *Pi and e In Binary - Exploring Binary.* http://www.exploringbinary.com/pi-and-e-in-binary/. Version: Mai 2011. – Viimati külastatud 2014-05-26

[11] STOLFI, J. ; FIGUEIREDO, L.H. d.: *An Introduction to Affine Arithmetic.* Version: 2003. http://www.sbmac.org.br/tema/seletas/docs/v4_3/101_01summary.pdf Viimati külastatud 2014-05-26

[12] Team, The Sage D.: *Fixed and Arbitrary Precision Numerical Fields — Sage Reference Manual v6.1.1: Fixed and Arbitrary Precision Numerical Fields*. `http://www.sagemath.org/doc/reference/rings_numerical/index.html`. – Viimati külastatud 2014-05-26

7 Lisad

1. **Sage kood:**

```
def random_matrix(RIF, Dim, Epsilon):
    A = Matrix(RIF,1,1,[0]);
    while A.is_singular():
        A = Matrix(RIF, Dim, Dim, [RIF.random_element() for i in range
            (Dim^2)]);
        for i in [0..Dim-1]: A[i,i]=sum(-A[i])+A[i,i]+RIF(Epsilon);
    return A;
def LU_factorization (LU_dim, LU_epsilon, LU_precision):

    ##### täpsuse määramine
    RIF = RealIntervalField(LU_precision);

        ##### ühik maatriks
    E = Matrix(RIF, LU_dim, LU_dim, [0 for i in (1..LU_dim*LU_dim)]);
    for i in [0..LU_dim-1]:E[i,i]=1;

        ##### koostame A, U, L
    A = random_matrix(RIF, LU_dim, LU_epsilon);
    x_teada = Matrix(RIF, LU_dim, 1, [RIF.random_element() for i in
        range(LU_dim)]);
    b = A * x_teada;

        U = copy(A);
    L = copy(E);

        ##### LU faktoriseering
    for i in [0..LU_dim-1]:
        if U[i,i] != 0:
            for j in [i+1..LU_dim-1]:
                L[j,i] = U[j,i] / U[i,i];
                U[j] = U[j] - U[i] * U[j,i] / U[i,i];

        y=L.inverse() * b;
    x=U.inverse() * y;
```

```
        return (x.column(0), x_teada.column(0));

def max_LU_fact(Dimension, Epsilon, Prec):
    return max(LU_factorization(Dimension, Epsilon, Prec)[0][i].
        diameter() for i in (0..(Dimension-1)));

def find_precision (Dimension, Diameter, Epsilon):
    for Prec in range(15,256):
        Matrix_d = max_LU_fact(Dimension, Epsilon, Prec);
        if Matrix_d <= Diameter:
            return Prec;
    return Infinity;

dimension = 8;
epsilon = 10^(-12);

# leiame e = 10^-12, dimension = 8 korral kui pikka mantissi osa on
    vaja,
# et LU-facotiseerimise tulemus oleks väiksem etteantud laiusest.
mantiss_data = [(log(diameter, 10), find_precision(dimension, diameter
    , epsilon)) for diameter in [10^(-j) for  j in [0..20]]];
Mantiss_plot=list_plot(mantiss_data, plotjoined=True);
Mantiss_plot.axes_labels(['$log_1$$_0(diameeter)$','$mantissi\ pikkus\
     bittides$']);
Mantiss_plot.show();

# Leiame diameetri sõltuvuse epsilonist dimension = 8, mantiss 53, 64,
     113 bitti,
# vastavalt 64, 80 ja 128 bitise IEEE 754 formaadi kohta
diameeter_53_data = [(log(eps, 10), log(max_LU_fact(dimension, eps,
    53),10)) for eps in [10^(-j) for  j in [0..10]]];
diameeter_64_data = [(log(eps, 10), log(max_LU_fact(dimension, eps,
    64), 10)) for eps in [10^(-j) for  j in [0..13]]];
diameeter_113_data = [(log(eps, 10), log(max_LU_fact(dimension, eps,
    113), 10)) for eps in [10^(-j) for  j in [0..20]]];
Diameeter_53_plot = list_plot(diameeter_53_data, plotjoined=True);
Diameeter_64_plot = list_plot(diameeter_64_data, plotjoined=True,
    color='red');
Diameeter_113_plot = list_plot(diameeter_113_data, plotjoined=True,
    color='yellow');
Diameeter_plot = Diameeter_53_plot + Diameeter_64_plot +
    Diameeter_113_plot;
```

```
Diameeter_plot.axes_labels(['$log_1$$_0(epsilon)$','$log_1$$_0(
    diameeter)$']);
Diameeter_plot.show();

A = LU_factorization(8, 10^(-1), 64);
Upper_plot = list_plot([A[0][x].upper() - A[1][x].upper() for x in
    range(8)], color='red' , plotjoined=True);
Lower_plot = list_plot([A[0][x].lower() - A[1][x].lower() for x in
    range(8)], color='blue' , plotjoined=True );
Erinevused_plot = Upper_plot + Lower_plot; Erinevused_plot.axes_labels
    (['$X/X\_teada$ element','tokete erinevus']);
Erinevused_plot.show();
Vahe_plot = list_plot([(A[0][x].upper() + A[0][x].lower() - A[1][x].
    upper() - A[1][x].lower())/2 for x in range(8)], color='blue',
    plotjoined=True);
Vahe_plot.axes_labels(['$X/X\_teada$ element','tokete erinevuste vahe
    ']);
Vahe_plot.show();
```

2. **Litsents**

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina **Karl Kruuse** (sünnikuupäev: 02.05.1987)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
   Ülevaade intervallaritmeetika meetodist
   mille juhendaja on professor Eero Vainikko,

   1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

   1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, 14.08.2014