Numeeriset menetelmät

Toni Karvonen
toni.karvonen@lut.fi

Lappeenrannan–Lahden teknillinen yliopisto LUT

27. helmikuuta 2026

Sisällys

1 Liukuluvut

Tämä osio käsittelee tapoja esittää reaalilukuja tietokoneessa. Koska reaalilukuja on ääretömän paljon, ei niitä kaikkia ole mahdollista esittää ja on välttämätöntä tehdä kompromisseja.

1.1 Kantalukujärjestelmät

Tutussa *kymmenjärjestelmässä* luvut esitetään kymmenen potenssien avulla. Esimerkiksi luku 1453,529 voidaan kirjoittaa muodossa

$$1453,529 = 1\cdot 10^3 + 4\cdot 10^2 + 5\cdot 10^1 + 3\cdot 10^0 + 5\cdot 10^{-1} + 2\cdot 10^{-2} + 9\cdot 10^{-3}.$$

Tällaisessa *paikkajärjestelmässä* numeron vaikutus luvun arvoon on sen arvon tulo valitun *kantaluvun* (kymmenjärjestelmässä kanta on 10) potenssin kanssa. Numeron sijainti määrää kantaluvun eksponentin. Sijainti lasketaan *desimaalierottimesta* siten, että vasemmalla puolella potenssit ovat epänegatiivisia ja oikealla puolella negatiivisia. Koska 5 on toinen numero erottimesta vasemmalle luvussa 1453,529, sen eksponentiksi tulee 1 (sillä ensimmäinen potenssi desimaalierottimen vasemmalla puolella on 0). Täten numeron 5 vaikutus tämän luvun arvoon on $5\cdot 10^1 = 50$. Vastaavasti koska 9 on kolmas numero desimaalierottimen oikealla puolella, sen eksponentti on -3 ja panos $9\cdot 10^{-3} = 0,009$.

Määritelmä 1.1 (Kantalukujärjestelmä)**.** Olkoon $b \geq 2$ kokonaisluku, jota kutsutaan *kantaluvuksi*. Kannassa b luvut kirjoitetaan muodossa

$$(a_n a_{n-1}\cdots a_0, c_1 c_2 \cdots)_b = a_n \cdot b^n + a_{n-1}\cdot b^{n-1} + \cdots + a_0 \cdot b^0 + c_1 \cdot b^{-1} + c_2 \cdot b^{-2} + \cdots,$$

missä *numerot* $a_0, \ldots, a_n$ ja $c_1, c_2, \ldots$ ovat kokonaislukuja välillä lukujen 0 ja $b-1$ välillä.

Alaindeksiä b ei yleensä käytetä kymmenjärjestelmässä (eli kun $b = 10$) muutoin kuin selvyyden vuoksi.

Esimerkki 1.2. Oktaaliluvun (eli $b = 8$) $(35,07)_8$ muuntaminen kymmenjärjestelmään tapahtuu seuraavasti:

$$(35,07)_8 = 3\cdot 8^1 + 5\cdot 8^0 + 0\cdot 8^{-1} + 7\cdot 8^{-2} = 3\cdot 8 + 5 + \frac{0}{8} + \frac{7}{8^2} = 29,109375.$$

Esimerkki 1.3. Heksadesimaalijärjestelmässä (eli kantaluku $b = 16$) kirjaimia A–F käytetään numeroiden 10–15 esittämiseen. Luvun $(\mathrm{AE},1\mathrm{B})_{16}$ muuntaminen kymmenjärjestelmään tapahtuu seuraavasti:

$$\begin{aligned}(\mathrm{AE},1\mathrm{B})_{16} &= 10\cdot 16^1 + 14\cdot 16^0 + 1\cdot 16^{-1} + 11\cdot 16^{-2}\\ &= 10\cdot 16 + 14 + \frac{1}{16} + \frac{11}{256}\\ &= 174,10546875.\end{aligned}$$

Esimerkki 1.4. Muuntaaksemme luvun $(220,1)_3$ kymmenjärjestelmään kirjoitamme

$$(220,1)_3 = 2\cdot 3^2 + 2\cdot 3^1 + 0\cdot 3^0 + 1\cdot 3^{-1} = 2\cdot 9 + 2\cdot 3 + \frac{1}{3} = 24,333\ldots$$

Huomaa, kantalukuesitys voi olla päättyvä yhdessä kannassa $[(220,1)_3]$ mutta päättymätön toisessa $(24,333\ldots)$.

Esimerkki 1.5. Kymmenjärjestelmän luku $33,25$ voidaan muuntaa binäärijärjestelmään (eli kantaluku $b=2$) seuraavasti:

$$\begin{aligned}(33,25)_{10} &= 32+1+\frac{1}{4} = 2\cdot 2^5+0\cdot 2^3+0\cdot 2^2+0\cdot 2^1+1\cdot 2^0+0\cdot 2^{-1}+1\cdot 2^{-2}\\ &= (100001,01)_2.\end{aligned}$$

Binäärijärjestelmä. Binäärijärjestelmän numerot (eli 0 ja 1) ovat binäärinumeroita eli *bittejä*. Esimerkiksi binääriluvussa $(110,1)_2 = 6,5$ on neljä bittiä. Suurin luku, joka voidaan esittää n bitillä, on

$$(\underbrace{11\cdots 11}_{n \text{ kertaa}})_2 = 1\cdot 2^{n-1}+1\cdot 2^{n-2}+\cdots+1\cdot 2^1+1\cdot 2^0 = 2^n-1. \tag{1.1}$$

1.2 Liukuluvut

Vaikka kaikki reaaliluvut pystyykin esittämään missä tahansa kannassa (esitys voi tosin olla päättymätön), käytännössä tämä on mahdotonta, sillä tietokoneiden muisti on rajallista. Kun lukuja toteutetaan tietokoneessa, on valittava, mitkä luvut esitetään ja miten. Tähän käytetään liukulukuesitystä.

Määritelmä 1.6 (Liukuluku)**.** *Liukuluku* $a\in\mathbb{R}$ kannassa b on muotoa

$$a = s\cdot c\cdot b^e,$$

missä $s\in\{-1,1\}$ on luvun *etumerkki*, $c\in\mathbb{R}$ *mantissa* ja $e\in\mathbb{Z}$ *eksponentti*.

Liukulukuesitys ei ole yksikäsitteinen. Esimerkiksi 123,45 voidaan kirjoittaa

$$12345\cdot 10^{-2} \quad \text{tai} \quad 1234,5\cdot 10^{-1}$$

tai äärettömän monella muulla tavalla kymmenkantaisena (eli $b=10$) liukulukuna. Jotta liukulukuja voidaan esittää tietokoneessa, on valittava äärellinen määrä mahdollisia mantissan ja eksponentin arvoja. Nämä valinnat määräävät syntyvän liukulukumuodon tarkkuuden ja muistinkulutuksen.

1.3 IEEE 754 -liukulukustandardi

IEEE-standardi liukulukulaskennalle (IEEE 754) määrittelee joukon liukulukumuotoja, joita käytetään käytännössä kaikessa laskennassa. Binääriset IEEE 754 -liukuluvut ovat muotoa

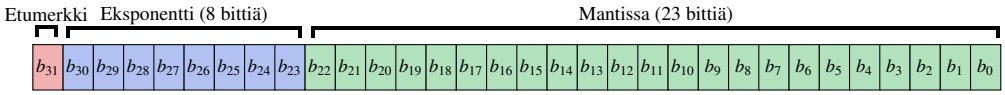
$$a = s\cdot(1+f)\cdot 2^{e-e_{\min}}, \tag{1.2}$$

missä $e_{\min}$, *eksponentin bias*, on positiivinen kokonaisluku ja $f\in(0,1)$ määrittää mantissan (kutsumme tätä lukua *mantissan murto-osaksi*). Sekä $f\in(0,1)$ että $e\in\mathbb{Z}_{\geq 0}$ tallennetaan binäärilukuina käyttäen valittua määrää bittejä. Toisin sanoen

$$f = (0.\bar{b}_1\bar{b}_2\cdots\bar{b}_p)_2 \quad \text{ja} \quad e = (\hat{b}_1\hat{b}_2\cdots\hat{b}_q)_2$$

joillekin positiivisille kokonaisluvuille p ja q sekä biteille $\bar{b}_i,\hat{b}_j\in\{0,1\}$. Etumerkki ilmoitetaan bitillä b_s muodossa $s=(-1)^{b_s}$, jolloin

$$b_s = 0 \implies s = 1 \qquad \text{ja} \qquad b_s = 1 \implies s = -1.$$



Kuva 1.1: 32-bittisten IEEE 754 -liukulukujen bittijako. Katso taulukko 1.1.

Tässä muodossa liukuluvun esittämiseen käytetään $n = p+q+1$ bittiä (p bittiä mantissalle, q eksponentille ja yksi etumerkille), jotka järjestetään seuraavasti:

$$(b_s \hat{b}_q \hat{b}_{q-1} \cdots \hat{b}_1 \bar{b}_p \bar{b}_{p-1} \cdots \bar{b}_1)_2 = (b_{p+q} b_{p+q-1} \cdots b_0)_2. \tag{1.3}$$

Kuva 1.1 havainnollistaa bittijakoa. IEEE 754 -esityksessä käytetään arvoa $e_{\min} = 2^{q-1} - 1$ kaavan (1.2) kokonaisluvulle $e_{\min}$ kaavassa (1.2). Tämä arvo määrää luvun 2 *varsinaisen eksponentin* $e - e_{\min} = e - 2^{q-1}$. Tästä seuraa, että varsinainen eksponentti vaihtelee [ks. (1.1)]

$$\text{luvusta} \quad e_{\min} = 2^{q-1} - 1 \quad \text{lukuun} \quad e_{\max} = 2^q - 1 - e_{\min} = 2(2^{q-1} - 1) = 2^{q-1}. \tag{1.4}$$

Esimerkki 1.7. Tarkastellaan IEEE 754 -muotoisia liukulukuja, joissa mantissalle on varattu $p = 3$ ja eksponentille $q = 2$ bittiä. Bittien kokonaismäärä on siis $n = p+q+1 = 6$. Tässä esitystavassa binääriluku $(101110)_2$ esittää muodon (1.2) mukaista liukulukua, jossa $s = (-1)^1 = -1$, $e = (01)_2 = 1$ ja $f = (0,110)_2 = 0,75$. Koska $e_{\min} = 2^{q-1} - 1 = 1$, $(101110)_2$ esittää kaavan (1.2) nojalla kymmenjärjestelmän luvun

$$s \cdot (1+f) \cdot 2^{e-e_{\min}} = (-1) \cdot (1+0,75) \cdot 2^{1-1} = -1,75.$$

IEEE 754 -standardin erikoistapaukset. Eksponenteilla $e = (0 \cdots 0)_2$ ja $e = (1 \cdots 1)_2$ on erityistulkinnat:

- Jos eksponentti on nolla, eli $e = (0 \cdots 0)_2$:
  - Jos *mantissan murto-osa on nolla*, eli $f = (0.0 \cdots 0)_2$, luku tulkitaan nollaksi. Etumerkkibitti määrää, onko nolla "positiivinen" vai "negatiivinen", mikä ei vaikuta useimpiin laskutoimituksiin. Toisin sanoen,

    $$\text{"positiivinen" nolla} = (10 \cdots 0)_2 \quad \text{ja} \quad \text{"negatiivinen" nolla} = (00 \cdots 0)_2.$$

  - Jos *mantissan murto-osa ei ole nolla*, liukulukua kutsutaan *alinormaaliksi* (engl. *subnormal*) ja se esittää kymmenjärjestelmän luvun

    $$a_{\text{subnormal}} = s \cdot f \cdot 2^{-e_{\min}+1}. \tag{1.5}$$

    Tämä poikkeaa kaavan (1.2) antamasta *normaalista* luvusta siten, että $1+f \in (1,2)$ korvautuu termillä $f \in (0,1)$. Alinormaaleilla luvuilla voidaan esittää pienempiä lukuja kuin pelkillä normaaleilla luvuilla olisi mahdollista.
- Jos eksponentti on suurin mahdollinen, eli $e = (1 \cdots 1)_2$:
  - Jos *mantissan murto-osa on nolla*, eli $f = (0.0 \cdots 0)_2$, luku tulkitaan etumerkkibitin mukaan positiiviseksi äärettömäksi (∞) tai negatiiviseksi äärettömäksi ($-\infty$).
  - Jos *murto-osa ei ole nolla*, luku tulkitaan *epäluvuksi* (NaN), eli arvo on määrittelemätön. Tämä voi tapahtua esimerkiksi laskettaessa $0/0$.

IEEE 754 -binäärimuodot. Taulukko 1.1 sisältää kolmen yleisimmän IEEE 754 -standardin binäärisen liukulukumuodon määritykset ja keskeiset ominaisuudet. Näihin viitataan tyypillisesti nimillä *puolitarkkuus* (engl. *half-precision*; 16 bittiä), *perustarkkuus* (engl. *float* tai *single-precision*; 32 bittiä) ja *kaksoistarkkuus* (engl. *double* tai *double-precision*; 64 bittiä).

Taulukko 1.1: IEEE 754 -standardin puolitarkkuus- (half), yksittäistarkkuus- (float) ja kaksoistarkkuuslukujen (double) keskeiset ominaisuudet. Huomaa, että jokaisessa muoto käyttää eksponentti- ja mantissabittien lisäksi yhden bitin etumerkin esittämiseen. *Kone-epsilon* kuvaa liukulukujärjestelmän tarkkuutta. Se on luvun 1 ja seuraavan sitä suuremman liukuluvun erotusta. Voidaan näyttää, että $\varepsilon = 2^{-p}$, missä p on mantissabittien lukumäärä. *Mantissan bittien määrä määrittää tarkkuuden ja eksponentin bittien määrä määrittää, kuinka pieniä ja suuria lukuja on mahdollista esittää.*

Ominaisuus	**Half (16 bittiä)**	**Float (32 bittiä)**	**Double (64 bittiä)**
Bittejä yhteensä	16	32	64
Bittejä eksponentissa (q)	5	8	11
Bittejä mantissassa (p)	10	23	52
Eksponentin bias ($e_{\min}$)	15	127	1023
Pienin pos. normaali luku	2^{-14}	2^{-126}	2^{-1022}
Pienin pos. alinormaali luku	2^{-24}	2^{-149}	2^{-1074}
Suurin äärellinen luku ($\approx$)	6.55×10^4	3.40×10^{38}	1.80×10^{308}
Kone-epsilon (ε)	2^{-10}	2^{-23}	2^{-52}

Esimerkki 1.8. Etsitään kymmenjärjestelmän luvun $a = -3,75$ perustarkkuusesitys (eli 32-bittinen esitys). Koska etumerkki $s = (-1)^{b_s}$, etumerkkibitti on yksi, eli $b_s = 1$. Kaavan (1.4) mukaan eksponentin bias on $e_{\min} = 2^{q-1} - 1 = 2^7 - 1 = 127$. Koska $1 + f \in (1,2)$ ja $(1+f)\cdot 2^1 \in (2,4)$, varsinaisen eksponentin täytyy olla 1, eli $1 = e - e_{\min} = e - 127$. Tästä saadaan $e = 128 = 2^7 = (10\cdots0)_2$, missä nollia on seitsemän. Yhtälöstä $3,75 = (1+f)\cdot 2$ ratkaistaan $f = \frac{1}{2}\cdot 3,75 - 1 = 0,875 = (0,111)_2$. Täten luvun $-3,75$ perustarkkuusesitys 32 bitillä on

$$11000000011100000000000000000000,$$

missä käytämme samoja värejä kuin kuvassa 1.1: etumerkki, eksponentti ja mantissan murto-osa, ja olemme lisänneet mantissan murto-osaan nollia varmistaaksemme, että esityksessä on 32 bittiä.

Esimerkki 1.9. Edetään toiseen suuntaan ja selvitetään, mitä kymmenjärjestelmän lukua a vastaa 32-bittinen perustarkkuusesitys

$$00001010001100000000000000000000.$$

Etumerkkibitti (sign) on nolla ($b_s = 0$), joten etumerkki on positiivinen, sillä $s = (-1)^{b_s} = (-1)^0 = 1$. Eksponentti on $e = (00010100)_2 = 2^4 + 2^2 = 20$. Mantissan murto-osa on $f = (0.01100000000000000000000)_2 = 2^{-2} + 2^{-3} = 0,375$. Siispä

$$a = s\cdot(1+f)\cdot 2^{e-e_{\min}} = 1\cdot(1+0,375)\cdot 2^{20-127} = 1,375\cdot 2^{-107} \approx 8,47\times 10^{-33}.$$

Esimerkki 1.10. Johdetaan joitakin taulukossa 1.1 lueteltuja ominaisuuksia perustarkkuusluvuille (32 bittiä). Kaavan (1.4) mukaan eksponentin bias on $e_{\min} = 2^{q-1} - 1 = 2^7 - 1 = 127$. Pienin mahdollinen positiivinen normaali luku on pienin positiivinen liukuluku, joka on esitettävissä muodossa (1.2). Täten meidän on määritettävä (a) mahdollisimman pieni mantissan murto-osa ja (b) mahdollisimman piene *positiivinen* eksponentti (luku on alinormaali,

Kuva 1.2: 16-bittisten IEEE 754 -liukulukujen bittijako. Katso taulukko 1.1.

jos eksponentti on nolla). Täten

$$f = (0.\underbrace{0\cdots0}_{\text{23 kertaa}})_2 = 0 \quad \text{ja} \quad e = (\underbrace{0\cdots0}_{\text{7 kertaa}}1)_2 = 1.$$

Siispä pienin positiivinen *normaali* perustarkkuusluku on

$$a = 1\cdot(1+f)\cdot 2^{e-e_{\min}} = 1\cdot(1+0)\cdot 2^{1-127} = 2^{-126} \approx 1{,}18\times 10^{-38}.$$

Pienimmän positiivisen *alinormaalin* luvun saamme asettamalla eksponentin nollaksi, valitsemalla pienimmän positiivisen mantissan murto-osan $f = (0.0\cdots01)_2 = 2^{-23}$ ja käyttämällä alinormaalin luvun määrittävää kaavaa (1.5). Näin saamme

$$a = 1\cdot f\cdot 2^{-127+1} = 2^{-23}\cdot 2^{-126} = 2^{-149} \approx 1{,}40\times 10^{-45}.$$

Suurimmalla äärellisellä perustarkkuusluvulla on mahdollisimman suuri mantissan murto-osa ja eksponentti, joka on yhtä pienempi kuin suurin mahdollinen (koska suurin eksponentti tulkitaan äärettömyydeksi tai epäluvuksi). Siispä

$$f = (0.\underbrace{1\cdots1}_{\text{23 kertaa}})_2 = \sum_{k=1}^{23} 2^{-k} = 1-2^{-23} \quad \text{ja} \quad e = (\underbrace{1\cdots1}_{\text{7 kertaa}}0)_2 = \sum_{k=1}^{7} 2^k = 254.$$

Näin saadaan suurimmaksi äärelliseksi perustarkkuusluvuksi

$$a = 1\cdot(1+f)\cdot 2^{e-e_{\min}} = (2-2^{-23})\cdot 2^{127} \approx 3{,}40\times 10^{38}.$$

Liukuluvut Pythonissa. Pythonissa liukuluvun 32-bittisen esityksen voi tulostaa seuraavasti:

```
import numpy as np
a = -3.75
res = np.binary_repr(np.float32(a).view(np.int32), width=32)
print(res)
```

Näiden koodirivien suorittaminen tulostaa `11000000011100000000000000000000`, mikä on täsmälleen esimerkissä 1.8 laskemamme esitys. Huomaa kuitenkin, että Pythonissa termillä “float” viitataan kaksoistarkkuuden, eli 64 bitin liukulukuihin.

1.4 Pyöristys ja tarkkuus

Liukulukulaskennalla on *äärellinen tarkkuus*. Toisin sanoen kaikkia reaalilukuja ei voida esittää täsmälleen ja laskutoimitusten tulokset voivat poiketa *täsmällisen laskennan* tuloksista. Esimerkiksi tuttu kymmenjärjestelmän luku $0{,}1$ ei ole esitettävissä täsmällisesti äärellisen monella bitillä, sillä sen binääriesitys on päättymätön:

$$(0,1)_{10} = (0,0001100110011001100110011...)_2.$$

Tästä seuraa, että tällaisia lukuja ei voida esittää täsmällisesti binäärisinä liukulukuina, vaan ne pyöristetään esitettävissä olevaan liukulukuun IEEE 754 -standardin määrittelemien pyöristys-

sääntöjen mukaisesti. Tähän törmää usein yksinkertaisissakin laskuissa. Esimerkiksi Pythonissa yksinkertainen kertolasku, jonka täsmällinen arvo on 653082380,34, ei tuotakaan täsmällistä arvoa:

```
>>> print(27869.01 * 23434)
653082380.3399999
```

Sama ilmiö havaitaan tulostettaessa 50 desimaalia esimerkiksi luvulle $a = 0,1$:

```
>>> a = 0.1
>>> print(f"{a:.{50}f}")
0.10000000000000000555111512312578270211815834045410
```

Tarkkuus. Liukulukujen *tarkkuus riippuu niiden kokoluokasta*. Tämä on sekä etu (liukuluvuilla voidaan esittää hyvin erikokoisia lukuja) että haitta (pyöristysvirheitä tulee väistämättä, kun operoidaan suurilla luvuilla). Esimerkiksi kaksoistarkkuusluvuilla (64 bittiä) voidaan esittää luvut välillä 1 ja 2 tarkkuudella 2^{-52}, eli kaikki luvut välillä $[1,2]$ pyöristyvät luvun 2^{-52} monikertoihin. Tarkkuus heikkenee lukujen kasvaessa: se on 2^{-51} lukujen välillä 2 ja 4 ja lopulta 2^{971} lukujen välillä 2^{1023} ja 2^{1024}. Tämän näkee havaitsee Pythonissa, joka käyttää oletuksena kaksoistarkkuuslukuja. Seuraavat laskutoimitukset palauttavat positiiviset arvot kuten niiden kuuluukin:

```
>>> b = 1.0
>>> a = 1.0 + 2.0**(-52)
>>> print(a - b)
2.220446049250313e-16
>>> b = 2.0**1023
>>> a = 2.0**1023 + 2.0**971
>>> print(a - b)
1.99584030953472e+292
```

Huomaa, että jälkimmäisessä laskussa on olennaista kirjoittaa "`2.0`" eikä "`2`", jotta Python tulkitsee annetut luvut liukuluvuiksi, ei kokonaisluvuiksi (joita se käsittelee eri tavoin). Jos kuitenkin teemme luvusta a hieman pienemmän, se pyöristyykin luvuksi b ja erotuksesta tulee nolla, mitä ei tapahtuisi täsmällisessä laskennassa:

```
>>> b = 1.0
>>> a = 1.0 + 0.5*2.0**(-52)
>>> print(a - b)
0.0
>>> b = 2.0**1023
>>> a = 2.0**1023 + 0.5*2.0**971
>>> print(a - b)
0.0
```

Alivuoto ja ylivuoto. Samankaltaisia ilmiöitä esiintyy laskettaessa luvuilla, jotka ovat pienempiä kuin *alivuototaso* (pienimmät positiiviset alinormaalit luvut taulukossa 1.1), jolloin laskutoimituksen tulos pyöristyy nollaan, tai suurempia kuin *ylivuototaso* (suurimmat äärelliset luvut taulukossa 1.1), jolloin tulos pyöristyy äärettömyyteen tai aiheuttaa virheen. 64-bittisillä kaksoistarkkuusluvuilla ali- ja ylivuototasot ovat 2^{-1074} ja noin $1,80 \times 10^{308}$.

1.5 Numeerinen vakaus

Liukulukujen luontaiset rajoitteet on otettava huomioon numeerisia algoritmeja suunniteltaessa. Algoritmin tai numeerisen menetelmän *numeerinen vakaus* (tai *stabiilisuus*) tarkoittaa sen kykyä "sietää" numeerisia virheitä. Erityistä varovaisuutta tulee noudattaa vähennettäessä toisiaan lähellä olevia liukulukuja. Mikäli mahdollista, tätä tulee välttää.

Esimerkki 1.11. Lauseke $\sqrt{1+x}-\sqrt{x} > 0$ on altis numeerisille kumoutumisille, kun x on suuri, koska tällöin sekä $\sqrt{1+x}$ että $\sqrt{x}$ ovat suuria ja hyvin lähellä toisiaan mutta kaukana lähimmistä esitettävissä olevista liukuluvuista (ks. osio 1.4). Esimerkiksi sijoitus $x = 2^{60}$ lausekkeeseen $\sqrt{1+x}-\sqrt{x}$ tuottaa arvon nolla Pythonin käyttämässä 64-bittisessä kaksoistarkkuuslaskennassa. Sen sijaan yhtäpitävä muoto

$$\sqrt{1+x}-\sqrt{x} = \frac{(\sqrt{1+x}-\sqrt{x})(\sqrt{1+x}+\sqrt{x})}{\sqrt{1+x}+\sqrt{x}} = \frac{1}{\sqrt{1+x}+\sqrt{x}},$$

joka käyttää identiteettiä $(a-b)(a+b) = a^2-b^2$, tuottaa positiivisen arvon jopa sijoitukselle $x = 2^{1023}$.

Esimerkki 1.12. Samanlainen ilmiö voi esiintyä laskettaessa a^2-b^2, kun a ja b ovat suuria. Laskua voidaan vakauttaa käyttämällä yhtälöä $(a-b)(a+b) = a^2-b^2$. Vaikka tämä uusi muoto sisältääkin vähennyslaskun $a-b$, vähennettävät luvut ovat pienempiä kuin alkuperäisessä lausekkesssa a^2-b^2 kun $a,b > 1$. Pyöristysvirhettäkin tulee näin vähemmän.

Seuraava koodinpätkä Pythonia havainnollistaa esimerkkiä 1.12:

```
>>> a = 2.0**50
>>> b = a - 1.0
>>> print(a**2 - b**2)
2251799813685248.0
>>> print((a - b)*(a + b))
2251799813685247.0
```

Kun $a = 2^{50}$ ja $b = 2^{50}-1$, lausekkeen a^2-b^2 täsmällinen arvo on 2251799813685247, jonka saimme liukuluvuilla käyttäen numeerisesti vakaampaa muotoa $(a-b)(a+b)$.

2 Virhe

Tämä osio käsittelee numeeristen virheiden luonnetta ja esitystapoja laskennassa.

2.1 Pyöristys- ja katkaisuvirhe

Numeeriset virheet johtuvat sekä äärellistarkkuisen liukulukulaskennan luontaisesta epätarkkuudesta että matemaattisten approksimaatioiden käytöstä.

Määritelmä 2.1 (Pyöristysvirhe)**.** *Pyöristysvirhe* on virhe, joka aiheutuu äärellistarkkuisen laskennane (esim. liukulukulaskennan) käytöstä.

Esimerkki 2.2. Osioissa 1.4 ja 1.5 käsitellyt pyöristys- ja tarkkuusongelmat ovat esimerkkejä pyöristysvirheistä.

Määritelmä 2.3 (Katkaisuvirhe)**.** *Katkaisuvirhe* on virhe, joka aiheutuu matemaattisen prosessin numeerisesta approksimaatiosta.

Esimerkki 2.4. Eksponenttivakion $e \approx 2.718$ sarjaesityksen

$$e = \sum_{k=0}^{\infty} \frac{1}{k!},$$

pitäisi olla tuttu matematiikan peruskursseilta. Tässä esityksessä $n! = 1 \cdots 2 \cdots (n-1) \cdot n$ on kokonaisluvun n kertoma, jolle on määritelty $0! = 1$. Vakion e approksimoimiseksi voidaan tämä sarja katkaista termin n kohdalla (tässä $n \in \{0,1,2,3\}$):

$$\begin{aligned}
e \approx e_0 &= \sum_{k=0}^{0} \frac{1}{k!} = \frac{1}{0!} = \frac{1}{1} = 1, \\
e \approx e_1 &= \sum_{k=0}^{1} \frac{1}{k!} = \frac{1}{0!} + \frac{1}{1!} = \frac{1}{1} + \frac{1}{1} = 2, \\
e \approx e_2 &= \sum_{k=0}^{2} \frac{1}{k!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} = \frac{1}{1} + \frac{1}{1} + \frac{1}{2} = 2{,}5, \\
e \approx e_3 &= \sum_{k=0}^{3} \frac{1}{k!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} = \frac{1}{1} + \frac{1}{1} + \frac{1}{2} + \frac{1}{2 \cdot 3} = 2{,}666\ldots
\end{aligned}$$

Jokainen erotuksista $e - e_n$ ($n \in \{0,1,2,3\}$) on katkaisuvirhe.

Katkaisuvirheellä ei kuitenkaan viitata ainoastaan ilmiselvään katkaisuun kuten esimerkissä 2.4, vaan yleensäkin numeerisiin approksimaatioihin.

Esimerkki 2.5. Palautetaan mieleen analyysistä, että funktion $f\colon \mathbb{R} \to \mathbb{R}$ derivaatta pisteessä $x_0 \in \mathbb{R}$ määritellään erotusosamäärän $[f(x_0+h) - f(x_0)]/h$ raja-arvona

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0+h) - f(x_0)}{h}.$$

Derivaattaa voi siis approksimoida laskemalla erotusosamäärän arvon jollakin pienellä positiivisella luvulla h. Esimerkiksi funktion $f(x) = x^2$ derivaatta kohdassa $x_0 = 1$ voidaan

approksimoida seuraavasti käyttäen lukua $h = 1/10$:

$$f'(x_0) \approx \frac{f(x_0+h)-f(x_0)}{h} = \frac{(1+1/10)^2-1^2}{1/10} = \frac{21}{10}.$$

Koska $f'(x) = 2x$, todellinen derivaatta on $f'(x_0) = f'(1) = 2$. Virhe

$$f'(x_0) - \frac{21}{10} = 2 - \frac{21}{10} = \frac{1}{10}$$

on katkaisuvirhe, vaikka mitään sarjaa ei katkaistakaan. Kuitenkin tässäkin tapauksessa voidaan erotusosamäärään perustuvaa approksimaatiota pitää eräänlaisena katkaisuna, sillä sitä käyttäen "katkaistaan raja-arvoprosessi" $h \to 0$ valitsemalla jokin pieni h raja-arvon itsensä käyttämisen sijaan.

2.2 Approksimaation tarkkuus

Määritelmä 2.6 (Approksimaation tarkkuus)**.** Olkoon $\tilde{x}$ approksimaatio luvulle x. Tällöin $\tilde{x}$ approksimoi lukua *x d:n desimaalin tarkkuudella*, jos

$$|x - \tilde{x}| < \frac{1}{2}10^{-d}.$$

Jos haluamme, että $\tilde{x}$ approksimoi lukua x kokonaislukujen tarkkuudella (eli nollan desimaalin tarkkuudella), täytyy päteä $|x - \tilde{x} < 0,5$. Tämä tarkoittaa, että lukujen x ja $\tilde{x}$ välinen etäisyys on nolla lähimpään kokonaislukuun pyöristettäessä. Tämä ei kuitenkaan tarkoita sitä, että molemmat luvut pyöristyvät samaan kokonaislukuun. Esimerkiksi jos $x = 0.51$ ja $\tilde{x} = 0.49$, niin $|0.51 - 0.49| = 0.02 < 0.5$, mutta $x \approx 1$ ja $\tilde{x} \approx 0$. Huomaa myös, että jos $\tilde{x}$ approksimoi lukua xd:n desimaalin tarkkuudella, se approksimoi sitä myös $(d-1)$:n desimaalin tarkkuudella ja niin edelleen. Olennaista on toki suurin mahdollinen desimaalitarkkuus, eli suurin d.

Esimerkki 2.7. Luku $3,14$ approksimoi piitä (π) kahden desimaalin tarkkuudella, sillä

$$|\pi - 3,14| = 0,00159\ldots = 0,159\ldots \cdot 10^{-2} < 0,5 \cdot 10^{-2}.$$

Luku $3,14$ approksimoi piitä myös yhden desimaalin tarkkuudella, sillä

$$|\pi - 3,14| = 0,00159\ldots = 0,0159\ldots \cdot 10^{-1} < 0,5 \cdot 10^{-1}.$$

Luku $3,14$ *ei* kuitenkaan approksimoi piitä kolmen desimaalin tarkkuudella, sillä

$$|\pi - 3.14| = 0,00159\ldots = 1,59\ldots \cdot 10^{-3} > 0,5 \cdot 10^{-3}.$$

2.3 Absoluuttinen ja suhteellinen virhe

Virhe voidaan mitata joko absoluuttisena tai suhteellisena.

Määritelmä 2.8 (Absoluuttinen ja suhteellinen virhe)**.** Olkoon $\tilde{x}$ luvun x approksimaatio. Tällöin

$$\textit{absoluuttinen virhe} = |x - \tilde{x}| \quad \text{ja} \quad \textit{suhteellinen virhe} = \frac{|x-\tilde{x}|}{|x|}.$$

Absoluuttinen virhe on luvun x ja sen approksimaation $\tilde{x}$ erotuksen itseisarvo. Suhteellinen virhe mittaa virhettä *suhteessa luvun x suuruuteen*. Huomaa, että suhteellista virhettä laskettaessa

jaetaan approksimoitavan luvun (eli x) itseisarvolla, ei approksimaation (eli $\tilde{x}$) itseisarvolla. Suhteellinen virhe ilmaistaan usein prosentteina.

Esimerkki 2.9. Jos lukua $x = \pi$ approksimoidaan kahden desimaalin tarkkuudella kuten esimerkissä 2.7 (vaikkapa $\tilde{x} = 3,14$), niin approksimaation absoluuttinen virhe on

$$|\pi - 3,14| = 0,00159\ldots$$

ja suhteellinen virhe

$$\frac{|\pi - 3,14|}{|\pi|} = 0,000507\ldots = 0.0507\ldots\,\%.$$

Suhteellinen virhe on usein näistä virheistä hyödyllisempi.

Esimerkki 2.10. Osiossa 1.4 havaitsimme, että Pythonin 64-bittisessä kaksoistarkkuuslaskennassa luku

$$x = 2^{1023} + 0.5 \cdot 2^{971} \quad \text{pyöristyy lukuun} \quad \tilde{x} = 2^{1023}.$$

Täten absoluuttinen pyöristysvirhe on

$$|x - \tilde{x}| = |2^{1023} + 0.5 \cdot 2^{971} - 2^{1023}| = 0.5 \cdot 2^{971} = 9.9792 \cdot 10^{291}.$$

Tämä verrattain suuri virhe on kuitenkin hyvin pieni suhteessa luvun x kokoon, sillä suhteellinen pyöristysvirhe on mitättömän pieni:

$$\frac{|x - \tilde{x}|}{|x|} = \frac{|2^{1023} + 0.5 \cdot 2^{971} - 2^{1023}|}{|2^{1023} + 0.5 \cdot 2^{971}|} \approx 1,11 \cdot 10^{-16} = 1,11 \cdot 10^{-14}\,\%.$$

2.4 Iso-O-notaatio

Kätevää iso-O-notaatiota käytetään approksimaatioiden asymptoottisen, eli raja-arvokäyttäytymisen, ilmaisemiseen.

Määritelmä 2.11 (Iso-O-notaatio funktioille)**.** Olkoon $x_0 \in (a,b)$ ja olkoot $f, g\colon (a,b) \to \mathbb{R}$ funktioita. Kirjoitamme

$$f(x) = O(g(x)) \quad \text{kun} \quad x \to x_0,$$

jos on olemassa $C \geq 0$ ja $\delta > 0$ siten, että $|f(x)| \leq Cg(x)$ kaikilla $x \in (x_0 - \delta, x_0 + \delta)$.

Palautetaan mieliin, että jono $(a_n)_{n=1}^{\infty} = (a_1, a_2, \ldots)$ on reaalilukujen järjestetty kokoelma.

Määritelmä 2.12 (Iso-O-notaatio jonoille)**.** Olkoot $(a_n)_{n=1}^{\infty}$ ja $(b_n)_{n=1}^{\infty}$ kaksi jonoa. Kirjoitamme

$$a_n = O(b_n) \quad \text{kun} \quad n \to \infty,$$

jos on olemassa $C \geq 0$ siten, että $|a_n| \leq Cb_n$ kaikilla $n \geq 1$.

Merkintä $O(g(x))$ luetaan esimerkiksi "suuruusluokkaa $g(x)$", "iso oo $g(x)$" tai "oo $g(x)$". Raja-arvo (joko $x \to x_0$ tai $n \to \infty$) jätetään usein merkitsemättä, mikäli se selviää asiayhteydestä. Tällä kurssilla käytämme iso-O-notaatiota kuvaamaan numeerisen approksimaation käyttäytymistä. Notaatio on kuitenkin hyvin monipuolinen ja sitä käytetään myös algoritmien aika- ja tilavaativuuden kuvaamiseen.[1] Jos $g(x) \to 0$ kun $x \to x_0$ (tai $b_n \to 0$ kun $n \to \infty$), väite $f(x) = O(g(x))$

[1] *Data Structures and Algorithms* (BM40A1500).

kertoo, kuinka nopeasti $f(x)$ lähestyy nollaa, kun $x \to x_0$. Tarkemmin sanottuna tämä tarkoittaa, että $f(x)$ lähestyy nollaa *vähintään* yhtä nopeasti kuin $g(x)$. Tällaisessa tilanteessa pyritään löytämään g, joka lähestyy nollaa mahdollisimman nopeasti siten, että $f(x) = O(g(x))$ on totta.

Esimerkki 2.13. Määritetään funktion $f(x) = x^2 + x^3$ suuruusluokka, kun $x \to 0$ (eli $x_0 = 0$). Määritelmän 2.11 mukaan meidän tulee löytää funktio g ja vakiot $C \geq 0$ ja $\delta > 0$ siten, että $|f(x)| \leq Cg(x)$ kaikille $x \in (-\delta, \delta)$. Valitaan $\delta = 1$ (mikä tahansa muukin $\delta \leq 1$ toimisi). Tällöin kaikilla $x \in (-1, 1)$ pätee

$$|f(x)| = x^2 + |x|^3 \leq x^2 + |x|^2 = 2x^2,$$

missä käytimme epäyhtälöä $|a|^n \leq |a|^m$, joka pätee kaikilla $|a| \leq 1$ ja $n \geq m$. Voimme siis valita $C = 2$ ja $g(x) = x^2$, mikä tarkoittaa, että $f(x)$ on suuruusluokkaa x^2 [$f(x) = O(x^2)$], kun $x \to 0$. Huomaa kuitenkin, että myös

$$|f(x)| = x^2 + |x|^3 \leq x^2 + |x|^2 \leq |x| + |x| = 2|x|,$$

joten olisimme voineet valita $g(x) = |x|$, jolloin $f(x) = O(|x|)$.

Esimerkissä 2.13 valinta $g(x) = x^2$ on parempi kuin $g(x) = |x|$, koska se kuvaa paremmin funktion f käyttäytymistä pisteen $x_0 = 0$ ympäristössä. Nimittäin, kun $x \to 0$,

$$f(x) = O(x^2) \quad \text{ja} \quad f(x) = O(|x|)$$

mutta

$$x^2 = O(|x|) \quad \text{ja} \quad |x| \neq O(x^2),$$

missä $|x| \neq O(x^2)$ tarkoittaa, että $|x|$ ei ole suuruusluokkaa x^2 kun $x \to 0$. Sanallisesti ilmaistuna nämä neljä iso-O-väitettä ovat

$$f(x) \text{ lähestyy nollaa vähintään yhtä nopeasti kuin } x^2$$
$$\text{ja} \quad f(x) \text{ lähestyy nollaa vähintään yhtä nopeasti kuin } |x|$$

mutta

$$x^2 \text{ lähestyy nollaa vähintään yhtä nopeasti kuin } |x|$$
$$\text{ja} \quad |x| \text{ lähestyy nollaa } \textit{hitaammin} \text{ kuin } x^2.$$

Kuva 2.1 havainnollistaa iso-O-notaatiota polynomeille pisteen $x_0 = 0$ läheisyydessä.

Huomautus 2.14. Notaation $f(x) = O(g(x))$ kanssa on syytä olla tarkkana. Koska yhtäsuuruus (“=”) on muualla matematiikassa vaihdannainen (eli $a = b$ on sama kuin $b = a$), $f(x) = O(g(x))$ antaa ymmärtää, että voitaisiin kirjoittaa $O(g(x)) = f(x)$, mitä on kuitenkin vaikea tulkita mielekkäästi. Täsmällisesti ilmaistuna $f(x) = O(g(x))$ kun $x \to x_0$ tarkoittaa, että *f kuuluu niiden funktioiden joukkoon, jotka ovat ylhäältä rajoitettuja jonkin vakion ja funktion g tulolla pisteen x_0 ympäristössä.* Merkintä on siis tulkittava vain kätevänä lyhennyksenä.

2.5 Iso-O-notaation ominaisuuksia

Seuraava propositio osoittaa, että iso-O-notaatio on *transitiivinen*. Käytännössä propositiossa ei ole kyse muusta kuin siitä perustuloksesta, että $a \leq c$ jos $a \leq b$ ja $b \leq c$.

Propositio 2.15. *Jos* $f(x) = O(g_1(x))$ *ja* $g_1(x) = O(g_2(x))$ *kun* $x \to x_0$*, niin*

$$f(x) = O(g_2(x)) \quad \text{kun} \quad x \to x_0.$$

Todistus. Määritelmän 2.11 nojalla on olemassa $C_1, C_2 \geq 0$ ja $\delta_1, \delta_2 > 0$ siten, että

$$|f(x)| \leq C_1 g_1(x) \quad \text{kaikilla} \quad x \in (x_0 - \delta_1, x_0 + \delta_1)$$

ja

$$|g_1(x)| \leq C_2 g_2(x) \quad \text{kaikilla} \quad x \in (x_0 - \delta_2, x_0 + \delta_2).$$

Nämä kaksi epäyhtälöä pätevät samanaikaisesti, kun $x \in (x_0 - \delta, x_0 + \delta)$, missä $\delta = \min\{\delta_1, \delta_2\}$. Täten kaikilla $x \in (x_0 - \delta, x_0 + \delta)$ pätee

$$|f(x)| \leq C_1 g_1(x) \leq C_1 |g_1(x)| \leq C_1 \cdot C_2 g_2(x),$$

joten voimme asettaa $g = g_2$, $C = C_1 \cdot C_2$ ja $\delta = \min\{\delta_1, \delta_2\}$ määritelmässä 2.11. □

Seuraava propositio osoittaa, ettei määritelmän 2.11 vakio C vaikuta suuruusluokkaan.

Propositio 2.16. *Jos* $f(x) = O(g_1(x))$ *kun* $x \to x_0$ *ja* $g_1(x) = C_{12} g_2(x)$ *jollakin* $C_{12} \geq 0$*, niin* $f(x) = O(g_2(x))$ *kun* $x \to x_0$*.*

Todistus. Määritelmän 2.11 nojalla on olemassa $C_1 \geq 0$ ja $\delta > 0$ siten, että $|f(x)| \leq C_1 g_1(x)$ kaikilla $x \in (x_0 - \delta, x_0 + \delta)$. Oletuksesta $g_1(x) = C_{12} g_2(x)$ seuraa, että kaikilla $x \in (x_0 - \delta, x_0 + \delta)$ pätee

$$|f(x)| \leq C_1 g_1(x) = C_1 \cdot C_{12} g_2(x),$$

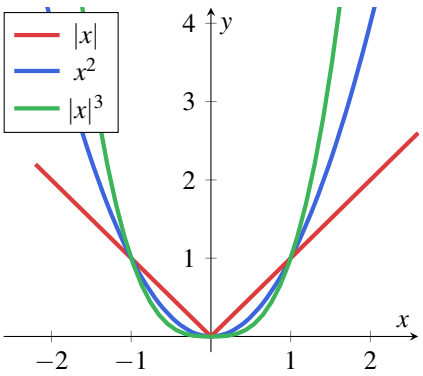
joten voimme valita $C = C_1 \cdot C_{12}$ ja $g = g_2$ määritelmässä 2.11. □

Propositioilla 2.15 ja 2.16 on luonnolliset vastineet jonoille (Määritelmä 2.12). Seuraava esimerkki havainnollistaa varsin tyypillistä iso-O-notaation käyttöä numeerisessa approksimaatiossa, jossa ollaan yleensä kiinnostuneita virheen käyttäytymisestä jonkin *diskretointiparametrin* h lähestyessä nollaa.

Esimerkki 2.17. Esimerkissä 2.5 approksimoimme funktion $f(x) = x^2$ derivaattaa kohdassa $x_0 = 1$ käyttäen erotusosamäärällä

$$q(h) = \frac{f(x_0 + h) - f(x_0)}{h}.$$

Lasketaan nyt tämän approksimaation absoluuttisen virheen suuruusluokka kun $h \to 0$.



Kuva 2.1: Funktiot $|x|$, x^2 ja $|x|^3$ pisteen $x_0 = 0$ läheisyydessä. Koska $|x|$ on x^2:n yläpuolella kun $x \in (-1, 1)$ ja x^2 on $|x|^3$:n yläpuolella, kuva havainnollistaa, että $x^2 = O(|x|)$ ja $|x|^3 = O(x^2)$, kun $x \to 0$. *Huomaa, että kuvaajien järjestys muuttuu, kun* $|x| > 1$. Jos tarkasteltaisiin pistettä x_0, jolle $|x_0| > 1$ (esim. $x_0 = 2$), iso-O-suhteet pitäisi kääntää: $|x| = O(x^2)$ ja $x^2 = O(|x|^3)$, kun $x \to 2$. Huomaa, että käytämme itseisarvoja $|x|$ ja $|x|^3$, sillä x ja x^3 ovat negatiivisia kun $x < 0$, mutta $f(x) = O(g(x))$ on väite ylärajasta itseisarvolle $|f(x)| \geq 0$, ei funktiolle $f(x)$.

Koska $f'(x) = 2x$ ja $x_0 = 1$, absoluuttinen virhe $\varepsilon(h)$ on

$$\begin{aligned}\varepsilon(h) = |f'(x_0) - q(h)| = \left|f'(x_0) - \frac{f(x_0+h)-f(x_0)}{h}\right| &= \left|2 - \frac{(1+h)^2-1}{h}\right| \\ &= \left|2 - \frac{2h+h^2}{h}\right| \\ &= |2-(2+h)| \\ &= |h|,\end{aligned}$$

missä käytimme identiteettiä $(a+b)^2 = a^2+2ab+b^2$. Täten $\varepsilon(h) = O(|h|)$, mikä tarkoittaa sitä, että erotusosamäärän avulla muodostetun approksimaation $f'(x_0) \approx q(h)$ virhe on suuruusluokaltaan *lineaarinen* kun $h \to 0$. Huomaa, että suhteellinen virhe $\rho(h)$ on samaa suuruusluokkaa, sillä

$$\rho(h) = \frac{|f'(x_0) - q(h)|}{|f'(x_0)|} = \frac{1}{|f'(x_0)|}\varepsilon(h),$$

missä $1/|f'(x_0)| = 1/2 > 0$. Propositiosta 2.16 seuraa nyt, että $\rho(h) = O(|h|)$.

Keskitymme pääosin virheisiin $\varepsilon(h)$, joilla on *polynominen suuruusluokka* kun $h \to 0$. Jos funktio on polynomi, pienimmän asteluvun termin kerroin määrää suuruusluokan.

Propositio 2.18. *Jos* $\varepsilon(h) = a_n h^n + a_{n+1}h^{n+1} + \cdots + a_{n+m}h^{n+m}$, *missä* $n, m \in \mathbb{N} \cup \{0\}$ *ja* $a_n, \ldots, a_{n+m} \in \mathbb{R}$, *niin*

$$\varepsilon(h) = O(|h|^n) \quad \textit{kun} \quad h \to 0.$$

Todistus. Huomaa, että $x_0 = 0$. Valitaan $\delta = 1$. Kun $h \in (x_0 - \delta, x_0 + \delta) = (-1, 1)$,

$$|\varepsilon(h)| = |a_n h^n + a_{n+1}h^{n+1} + \cdots + a_{n+m}h^{n+m}| \leq |a_n||h|^n + \cdots + |a_{n+m}||h|^{n+m},$$

missä käytimme kolmioepäyhtälöä $|a+b| \leq |a| + |b|$. Koska $|h|^{n+m} \leq |h|^n$ kun $h \in (-1,1)$, saamme epäyhtälön

$$|\varepsilon(h)| \leq (|a_n| + \cdots + |a_{n+m}|)|h|^n,$$

missä vakio $C = |a_n| + \cdots + |a_{n+m}|$ on epänegatiivinen (eli ≥ 0). Määritelmästä 2.11 seuraa nyt, että $\varepsilon(h) = O(|h|^n)$. □

Emme ole vielä käsitelleet määritelmää 2.12, eli iso-O-notaatiota jonoille. Tällä kurssilla käytämme tätä notaatiota kuvaamaan numeeristen menetelmien virhettä kun n, *diskretisointipisteiden* tai *aika-askelten* lukumäärä, kasvaa.

Esimerkki 2.19. Palataan esimerkkiin 2.4, jossa approksimoimme eksponenttivakiota $e \approx 2.718$ katkaistulla sarjakehitelmällä $e = \sum_{k=0}^{\infty} 1/k!$ n:

$$e \approx e_n = \sum_{k=0}^{n} \frac{1}{k!}.$$

Määritetään tämän approksimaation suuruusluokka (eli virheen $\varepsilon_n = |e - e_n|$ suuruusluokka), kun $n \to \infty$. Saamme

$$\varepsilon_n = |e - e_n| = \left|\sum_{k=0}^{\infty}\frac{1}{k!} - \sum_{k=0}^{n}\frac{1}{k!}\right| = \left|\sum_{k=n+1}^{\infty}\frac{1}{k!}\right| = \sum_{k=n+1}^{\infty}\frac{1}{k!}. \tag{2.1}$$

Mille tahansa $m \geq 1$ voimme kirjoittaa

$$(n+m)! = 1 \cdot 2 \cdots (n+1) \cdot (n+2) \cdots (n+m) = (n+1)! \cdot (n+2) \cdots (n+m), \tag{2.2}$$

missä tulo $(n+2)\cdots(n+m)$ tulkitaan ykköseksi, jos $m = 1$. Sijoittamalla yhtälön (2.2) yhtälöön (2.1) saamme

$$\varepsilon_n = \sum_{k=n+1}^{\infty} \frac{1}{k!} = \sum_{m=1}^{\infty} \frac{1}{(n+m)!} = \frac{1}{(n+1)!}\left(1 + \sum_{m=2}^{\infty} \frac{1}{(n+2)\cdots(n+m)}\right).$$

Viimeinen sarja suppenee (ohitamme todistuksen), mistä seuraa, että voimme valita

$$a_n = \varepsilon_n, \quad b_n = \frac{1}{(n+1)!} \quad \text{ja} \quad C = 1 + \sum_{m=2}^{\infty} \frac{1}{(n+2)\cdots(n+m)}$$

määritelmässä 2.12. Täten

$$\varepsilon_n = |e - e_n| = O\left(\frac{1}{(n+1)!}\right).$$

3 Numeeristen menetelmien rakentaminen

Tämä osio käsittelee numeeristen menetelmien rakentamisen ja implementoinni peruskäsitteistöä ja -työkaluja.

3.1 Taylorin polynomi

Monet numeeriset menetelmät hyödyntävät sileän (eli riittävän monta kertaa derivoituvan) funktion Taylorin polynomeja.

Määritelmä 3.1 (Taylorin polynomit)**.** Olkoon $f\colon (a,b) \to \mathbb{R}$ funktio, joka on n kertaa derivoituva pisteessä $x_0 \in (a,b)$. Funktion f *kertaluvun n Taylorin polynomi* pisteessä x_0 on

$$P_n(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{f''(x_0)}{2!}(x-x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x-x_0)^n. \tag{3.1}$$

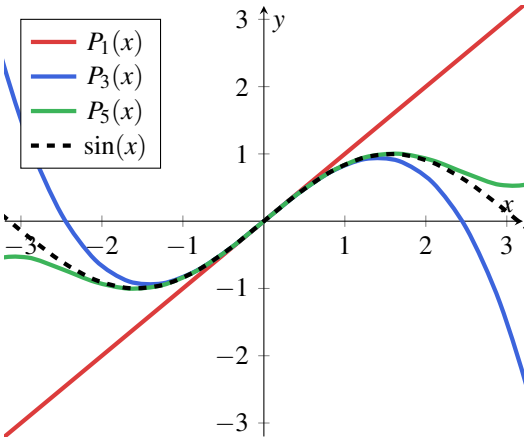
Palautetaan mieliin, että $f^{(n)}(x)$ tarkoittaa funktion f kertaluvun n derivaattaa pisteessä x, eli

$$f^{(n)}(x) = \frac{\mathrm{d}^n}{\mathrm{d}x^n} f(x).$$

Lisäksi nollas derivaatta on funktio itse: $f^{(0)}(x) = f(x)$. Taylorin polynomit voidaan siis kirjoittaa käyttäen summamerkintää:

$$P_n(x) = \sum_{k=0}^{n} \frac{f^{(n)}(x_0)}{n!}(x-x_0)^n.$$

Esimerkki 3.2. Muodostetaan kertalukujen $n \in \{1,3,5\}$ Taylorin polynomit funktiolle $f(x) = \sin(x)$ pisteessä $x_0 = 0$. Tarvitsemme tätä varten funktion f derivaattojen arvot viidenteen kertalukuun saakka pisteessä $x_0 = 0$. Muistamalla, että funktion $\sin(x)$ derivaatta



Kuva 3.1: Funktio $f(x) = \sin(x)$ sekä sen Taylorin polynomit P_1, P_3 ja P_5, joiden kertaluvut ovat $n \in \{1,3,5\}$. Taylorin polynomit määritellään yhtälössä (3.1) ja lasketaan sinifunktiolle esimerkissä 3.2. Kuvasta havaitsemme, että Taylorin polynomin kertaluvun kasvattaminen parantaa approksimaation $f(x) \approx P_n(x)$ tarkkuutta. Polynomi P_1 on kelvollinen approksimaatio kun $|x| < 1$, kun taas P_5 toimii hyvin kun $|x| < 2.5$.

on $\cos(x)$ ja funktion $\cos(x)$ derivaatta on $-\sin(x)$, saamme

$$\begin{aligned}
f^{(0)}(x) = f(x) = \sin(x) &\implies f^{(0)}(0) = 0,\\
f'(x) = \cos(x) &\implies f'(0) = 1,\\
f''(x) = -\sin(x) &\implies f''(0) = 0,\\
f^{(3)}(x) = -\cos(x) &\implies f^{(3)}(0) = -1,\\
f^{(4)}(x) = \sin(x) &\implies f^{(4)}(0) = 0,\\
f^{(5)}(x) = \cos(x) &\implies f^{(5)}(0) = 1.
\end{aligned}$$

Sijoittamalla nämä arvot kaavaan (3.1) saamme Taylorin polynomit

$$\begin{aligned}
P_1(x) &= f(x_0) + f'(x_0)(x - x_0) = f(0) + f'(0)x = x,\\
P_3(x) &= f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \frac{f^{(3)}(0)}{3!}x^3 = x - \frac{1}{6}x^3,\\
P_5(x) &= f^{(0)}(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \frac{f^{(3)}(0)}{3!}x^3 + + \frac{f^{(4)}(0)}{4!}x^4 + \frac{f^{(5}(0)}{5!}x^5\\
&= x - \frac{1}{6}x^3 + \frac{1}{120}x^5.
\end{aligned}$$

Näet nämä polynomit kuvassa 3.1.

Esimerkki 3.2 ja kuva 3.1 viittaavat siihen, että Taylorin polynomin kertaluvun tuottaa paremman approksimaation funktiolle f. Matemaattisesta analyysistä tuttu Taylorin lause selittää tämän havainnon.

Lause 3.3 (Taylorin lause)**.** *Olkoon $f\colon [a,b] \to \mathbb{R}$ jatkuva välillä $[a,b]$ ja $n+1$ kertaa derivoituva välillä (a,b). Olkoon $x_0 \in (a,b)$. Tällöin jokaisella $x \in [a,b]$ pätee*

$$f(x) - P_n(x) = R_{n+1}(x),$$

missä jäännöstermi R_{n+1} *on*

$$R_{n+1}(x) = \frac{f^{(n+1)}(c_x)}{(n+1)!}(x - x_0)^{n+1}$$

jollakin pisteellä c_x, joka on pisteiden x_0 ja x välillä.

Luvun c_x arvo Taylorin lauseessa riippuu pisteestä x ja kertaluvusta n. Iso-O-notaatiotalla (osio 2.4) voimme kirjoittaa Taylorin lauseen yksinkertaisemmin, mikäli funktion f kertaluvun $(n+1)$ derivaatta on rajoitettu.

Korollaari 3.4. *Olkoon $f\colon [a,b] \to \mathbb{R}$ jatkuva välillä $[a,b]$ ja $n+1$ kertaa derivoituva välillä (a,b). Olkoon $x_0 \in (a,b)$. Jos on olemassa vakio $L \geq 0$, jolla $|f^{(n+1)}(x)| \leq L$ kaikilla $x \in (a,b)$, niin*

$$R_{n+1}(x) = O(|x - x_0|^{n+1}) \quad \textit{kun} \quad x \to x_0. \tag{3.2}$$

Todistus. Taylorin lause ja oletus $|f^{(n+1)}(x)| \leq L$ kaikilla $x \in (a,b)$ antavat

$$|R_{n+1}(x)| = \left|\frac{f^{(n+1)}(c_x)}{(n+1)!}(x - x_0)^{n+1}\right| = \frac{|f^{(n+1)}(c_x)|}{(n+1)!}|x - x_0|^{n+1} \leq \frac{L}{(n+1)!}|x - x_0|^{n+1}$$

kaikilla $x \in (a,b)$. Koska $x_0 \in (a,b)$, on olemassa $\delta > 0$ siten, että $(x_0 - \delta, x + \delta) \subset (a,b)$, jolloin yllä oleva epäyhtälö pätee kaikilla $x \in (x_0 - \delta, x + \delta)$. Täten voimme valita $f(x) = R_{n+1}(x)$, $g(x) = |x - x_0|^{n+1}$ ja $C = L/(n+1)!$ määritelmässä 2.11. Tämä todistaa väitteen (3.2). □

Valitsemalla $x = x_0 + h$ (jolloin $x - x_0 = h$) voidaan (3.2) kirjoittaa muotoon

$$R_{n+1}(x_0 + h) = O(|h|^{n+1}) \quad \text{kun} \quad h \to 0,$$

mikä korostaa jäännöstermin pienenmistä tarkasteltaessa pisteen x_0 läheisyydessä olevia pisteitä (eli tapausta $h \to 0$).

Esimerkki 3.5. On tärkeää ymmärtää, että jäännöstermi R_{n+1} on pieni vain pisteen x_0 läheisyydessä. Tarkastellaan esimerkiksi kertaluvun $n = 2$ Taylorin polynomia pisteessä $x_0 = 1$ funktiolle $f(x) = x^4 + 2x^2$. Koska f on määritelty koko reaaliakselilla $\mathbb{R}$, voimme valita minkä tahansa välin, joka sisältää x_0:n, ja käyttää sitä korollaarissa 3.4. Valitaan $[a,b] = [0,2]$. Tällöin (koska $n+1 = 3$)

$$|f^{(3)}(x)| = \left|\frac{\mathrm{d}^2}{\mathrm{d}x^2}(4x^3 + 4x)\right| = \left|\frac{\mathrm{d}}{\mathrm{d}x}(12x^2 + 4)\right| = |24x| \leq 48 \quad \text{kaikilla} \quad x \in [0,2].$$

Siksi voimme soveltaa korollaaria 3.4 arvoilla $n = 2$, $x_0 = 1$ ja $L = 48$ ja päätellä, että

$$f(x) - P_2(x) = R_3(x) = O(|x-1|^3) \quad \text{kun} \quad x \to 1,$$

mistä seuraa, että kertaluvun kaksi Taylorin polynomin P_2 virhe on suuruusluokaltaan *kuutiollinen* kun $x \to x_0 = 1$. Tilanne muuttuu kuitenkin täysin, kun liikutaan poispäin pisteestä $x_0 = 1$. Koska $n = 2$, nähdään Taylorin polynomin määritelmästä (3.1), että P_2 on *toisen* asteen polynomi, eli $P_2(x) = \alpha x^2 + \beta x + \gamma$ joillakin $\alpha, \beta, \gamma \in \mathbb{R}$. Koska $f(x) = x^4 + 2x^2$, tästä seuraa, että

$$f(x) - P_2(x) = x^4 + (2 - \alpha)x^2 - \beta x - \gamma,$$

joka on *neljännen* asteen polynomi, korkeimman asteen termin kerroin on positiivinen (eli asteen 4 termin kerroin on 1). Tästä seuraa, että

$$f(x) - P_2(x) \to \infty \quad \text{kun} \quad x \to \infty \text{ tai } x \to -\infty.$$

Näin ollen virheestä voidaan tehdä mielivaltaisen suuri valitsemalla x, joka on riittävän kaukana pisteestä $x_0 = 1$.

3.2 Linearisointi

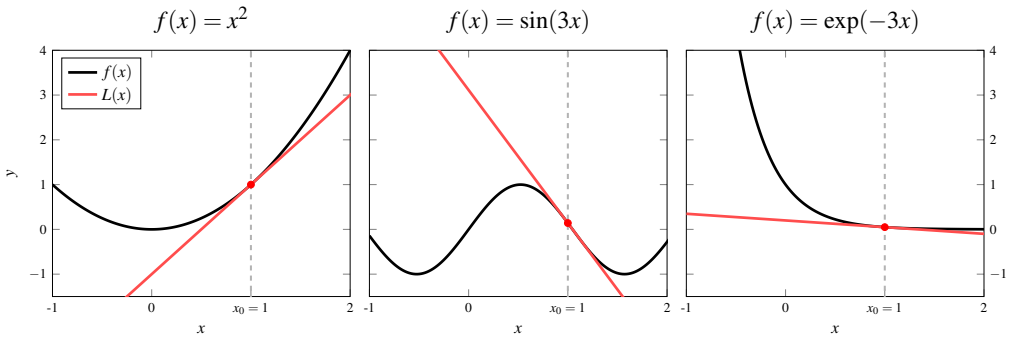
Taylorin polynomi (Määritelmä 3.1) tapauksessa $n = 1$ on eräs tärkeimmistä työkaluista numeeristen menetelmien kehityksessä.

Määritelmä 3.6 (Lineaarinen approksimaatio)**.** Olkoon $f\colon (a,b) \to \mathbb{R}$ derivoituva pisteessä $x_0 \in (a,b)$. Funktion f *lineaarinen approksimaatio* pisteessä x_0 on lineaarinen polynomi

$$L(x) = P_1(x) = f(x_0) + f'(x_0)(x - x_0). \tag{3.3}$$

Lineaarisen approksimaation L laskemista ja käyttämistä funktion f approksimointiin kutsutaan *linearisoinniksi*.

Lineaarinen approksimaatio on korkeamman kertaluvun Taylorin polynomeja helpompi laskea ja käyttää, sillä sen laskemiseen tarvitaan ainoastaan $f(x_0)$ ja $f'(x_0)$. Sovelluksissa korkeamman kertaluvun derivaattojen laskeminen on usein hankalaa tai jopa mahdotonta. Lisäksi linearisointi yleistyy helposti useamman muuttujan funktioihin $f\colon \mathbb{R}^d \to \mathbb{R}$ *gradientin* avulla ja tuottaa numeerisia menetelmiä, joiden implementointi on suoraviivaista ja joiden toimintaa on helppo ymmärtää.



Kuva 3.2: Lineaariset approksimaatiot (punainen) kolmelle funktiolle $f(x) = x^2$, $f(x) = \sin(3x)$ ja $f(x) = \exp(-3x)$ pisteessä $x_0 = 1$. Havaitaan, että lineaariset approksimaatiot ovat *tangenttisuoria* pisteessä $x_0 = 0$. Toisin sanoen lineaarinen approksimaatio on suora, jonka kulmakerroin on funktion f derivaatta pisteessä x_0.

Esimerkki 3.7. Muodostetaan lineaariset approksimaatiot funktioille

$$f_1(x) = -x^2, \quad f_2(x) = \sin(\tfrac{\pi}{2}x) \quad \text{ja} \quad f_3(x) = e^x - x^2$$

pisteessä $x_0 = 1$. Funktiolle f_1 saadaan

$$f_1(1) = -1 \text{ ja } f_1'(1) = -2 \quad \Longrightarrow \quad L(x) = f(1) + f'(1)(x-1) = -2x + 1.$$

Funktiolle f_2 saadaan [muista trigonometriasta $\sin(\frac{\pi}{2}) = 1$]

$$f_2(1) = 1 \text{ ja } f_2'(1) = 0 \quad \Longrightarrow \quad L(x) = f(1) + f'(1)(x-1) = 1.$$

Funktiolle f_3 saadaan

$$f_3(1) = e - 1 \text{ ja } f_3'(1) = e - 2 \quad \Longrightarrow \quad L(x) = f(1) + f'(1)(x-1) = (e-2)x + 1.$$

Kuva 3.2 näyttää lineaariset approksimaatiot pisteessä $x_0 = 1$ kolmelle samankaltaiselle funktiolle.

Esimerkki 3.7 ja kuva 3.2 osoittavat, että linearisointi vastaa sellaisen suoran piirtämistä, joka (a) kulkee pisteen $(x_0, f(x_0))$ kautta ja (b) jonka kulmakerroin on $f'(x_0)$.

Propositio 3.8. *Lineaarinen approksimaatio on ainoa suora, joka kulkee pisteen $(x_0, f(x_0))$ kautta ja jonka kulmakerroin on $f'(x_0)$.*

Todistus. Jokaisen suoran voi esittää yhtälöllä $y = \alpha x + \beta$ joillakin $\alpha, \beta \in \mathbb{R}$, missä α on suoran kulmakerroin. Se, että suora kulkee pisteen $(x_0, f(x_0))$ kautta, tarkoittaa että $f(x_0) = \alpha x_0 + \beta$. Se, että suoran kulmakerroin on $f'(x_0)$, tarkoittaa että $\alpha = f'(x_0)$. Siispä

$$f(x_0) = \alpha x_0 + \beta = f'(x_0)x_0 + \beta \quad \Longrightarrow \quad \beta = f(x_0) - f'(x_0)x_0.$$

Täten kaikilla väitteen suorilla on $\alpha = f'(x_0)$ ja $\beta = f(x_0) - f'(x_0)x_0$. Toisin sanoen on täsmälleen yksi suora, joka kulkee pisteen $(x_0, f(x_0))$ kautta ja jonka kulmakerroin on $f'(x_0)$. Tämän suoran yhtälö on

$$y = \alpha x + \beta = f'(x_0)x + f(x_0) - f'(x_0)x_0 = f(x_0) + f'(x_0)(x - x_0),$$

joka on täsmälleen kaavan (3.3) lineaarinen approksimaatio. □

Kuvasta 3.2 näemme, että linearisointi antaa hyvän approksimaation kun x on lähellä pistettä x_0, mutta approksimaation tarkkuus voi heikentyä nopeasti, kun liikutaan kauemmas tästä pisteestä. Seuraavan tuloksen mukaan lineaarisen approksimaation virhe on *neliöllistä* kertaluokkaa kun $x \to x_0$, jos f on kahdesti derivoituva.

Propositio 3.9. *Olkoon $f\colon [a,b] \to \mathbb{R}$ jatkuva välillä $[a,b]$ ja kahdesti derivoituva välillä (a,b). Olkoon $x_0 \in (a,b)$. Jos on olemassa vakio $L \geq 0$, jolla $|f''(x)| \leq L$ kaikilla $x \in (a,b)$, niin*

$$f(x) - L(x) = O(|x - x_0|^2) \quad \textit{kun} \quad x \to x_0.$$

Todistus. Koska $L(x) = P_1(x)$, väite seuraa asettamalla $n = 1$ korollaarissa 3.4. □

3.3 Taulukko-ohjelmointi

Taulukko-ohjelmointi, eli *vektorisointi*, tarkoittaa ohjelmointityyliä, jossa matemaattisia operaatioita sovelletaan kerralla kaikkiin taulokon arvoihin (vektoriin) sen sijaan, että niitä sovellettaisiin jokaiseen arvoon erikseen (eli silmukalla). Taulukko-ohjelmoinnilla toteutetut algoritmit vat yleensä *helppolukuisempia* kuin vastaavat silmukoin toteutetut algoritmit. Monissa korkean abstraktion kielissä, kuten Pythonissa ja MATLABissa, taulukko-ohjelmointi on myös *huomattavasti nopeampaa* kuin silmukat ja sitä kannattaa hyödyntää aina. Oletetaan, että haluamme korottaa neliöön (eli laskea x^2) kymmenen miljoonaa lukua x välillä nollasta yhteen käyttäen Pythonia. Taulukko-ohjelmointiin käytämme Pythonin `numpy`-kirjastoa.

```
>>> import numpy as np
>>> import time
>>> # an array of ten million numbers between 0 and 1
>>> a = np.linspace(start=0, stop=1, num=int(1e7))
>>> # use loop to square each number
>>> start1 = time.time()
>>> b1 = [x**2 for x in a]
>>> end1 = time.time()
>>> # square each number using array programming
>>> start2 = time.time()
>>> b2 = a**2
>>> end2 = time.time()
>>> # print elapsed times
>>> print(end1 - start1)
0.5299568176269531
>>> print(end2 - start2)
0.007016181945800781
```

Vaikka kulunut aika riippuu tietokoneesta ja ajokerrasta, taulukko-ohjelmointi on aina nopeampaa.

Esimerkki 3.10. Monte Carlo -integrointi on yksinkertainen menetelmä funktion integraalin approksimoimiseen. Oletetaan, että haluamme approksimoida jonkin funktion $f\colon [0,1] \to \mathbb{R}$ integraalia $\int_0^1 f(x)\,\mathrm{d}x$. Monte Carlo -integroinnissa arvotaan n lukua $x_1, \ldots, x_n$ tasaisesti välillä $[0,1]$ ja approksimoidaan integraalia funktion näiden pisteiden avulla laskettulla keskiarvolla:

$$\int_0^1 f(x)\,\mathrm{d}x \approx \frac{1}{n}\sum_{i=1}^{n} f(x_i). \tag{3.4}$$

Taulukko-ohjelmointia tukevissa ohjelmointikielissä Monte Carlo -approksimaation (3.4)

laskeminen onnistuu ilman yhtäkään silmukkaa.

Pythonissa approksimaatio ($n = 200$) funktion $f(x) = x^{3.4} + \sin(x)$ integraalille voidaan laskea esimerkiksi näin:

```
>>> import numpy as np
>>> rng = np.random.default_rng()
>>> # draw n = 200 random numbers on [0, 1]
>>> n = 200
>>> samples = rng.random(size=20)
>>> # evaluate f at these point and compute average
>>> evals = np.power(samples, 3.4) + np.sin(samples)
>>> mc_approx = np.sum(evals) / n
>>> print(mc_approx)
0.692222065735476
```

Lopputulos vaihtelee ajokerrasta toiseen, koska 200 pistettä x_i arvotaan satunnaisesti. MATLABissa tämän voi tehdä seuraavasti:

```
>>> # draw n = 200 random numbers on [0, 1]
>>> n = 200;
>>> samples = rand(n, 1);
>>> # evaluate f at these point and compute average
>>> evals = samples.^(3.4) + sin(samples);
>>> mc_approx = sum(evals) / n
mc_approx =

   0.6686
```

Tässä käytimme MATLABin pistepotenssia (eli alkioittaista potenssi), joka laskee potenssin $x^{3.4}$ jokaiselle satunnaisluvulle x.

4 Numeerinen integrointi

4.1 Keskipistesääntö

4.2 Puolisuunnikassääntö

4.3 Simpsonin sääntö

4.4 Virherajoja

5 Yhtälöryhmien ratkaiseminen

5.1 Newtonin menetelmä

5.2 Kiintopistemenetelmä

5.3 Suorat ja iteratiiviset menetelmät yhtälöryhmille

5.4 Newtonin menetelmä yhtälöryhmille

5.5 Jacobin menetelmä yhtälöryhmille

6 Differentiaaliyhtälöt

6.1 Eulerin menetelmä

6.2 Runge–Kutta-menetelmät

7 Optimointi

7.1 Optimoinnin peruskäsitteitä

7.2 Gradienttimenetelmä

7.3 Automaattinen derivointi