# Fel och datoraritmetik

**Avrundningsfel och flyttalssystem**
Från labben:

- diskretiseringsfel, avrundningsfel
- Beräkningen $A^{-1}A$ blev inte riktigt enhetsmatrisen
- olika sätt att mäta fel (rel fel, absolut fel)
- begrepp $\epsilon_M$, Inf, NaN, overflow, underflow

**Avrundningsfel och flyttalssystem**
Exempel)

| Uttryck | blir exakt | i MATLAB |
|---|---|---|
| $cos(\pi/2)$ | 0 | 6.1232e-017 |
| $0.42 - 0.5 + 0.08$ | 0 | -1.3878e-017 |
| $0.08 + 0.42 - 0.5$ | 0 | 0 |
| $A^{-1} \cdot A$ | Enhetsmatrisen | Se lab |

Varför blir det inte alltid exakt rätt svar när man arbetar i MATLAB?
Testa gärna även olika beräkningar på din miniräknare.

**Avrundningsfel och flyttalssystem**
Först hur man mäter fel:
Det exakta talet betecknas $x$ och $\hat{x}$ samma tal som innehåller något fel t ex fel från avrundningar eller mätfel.

- Absolut fel $|x - \hat{x}|$
- Relativt fel $\frac{|x-\hat{x}|}{|x|}$

Om $x$ är en vektor blir det istället

- Absolut fel $||x - \hat{x}||$
- Relativt fel $\frac{||x-\hat{x}||}{||x||}$

$||\cdot||$ kallas för norm. Mer om detta senare.

**Avrundningsfel och flyttalssystem**
**Exempel (fr labben)**

- Du köper en varmkorv en lördagkväll. Den kostar 15 kr, men av misstag ger du 20 kr.
  Absolut fel: $|15 - 20| = 5$
  Relativt fel: $\left|\frac{|15-20|}{|15|}\right| \approx 0.333 = 33.3\%$

- Du köper en ny bil för 299 995 kr. Du betalar 300 000 och bryr dig inte om växeln tillbaka.
  Absolut fel: $|299995 - 300000| = 5$
  Relativt fel: $\left|\frac{|299995-300000|}{|299995|}\right| \approx 0.0000166 \approx 0.0017\%$

Ofta uppfattar man det relativa felet mer korrekt (inte alltid). I båda fallen förlorar man lika mycket, men det känns sannolikt mindre i det andra fallet.

**Avrundningsfel och flyttalssystem**
Representation av tal i dator:
Alla tal lagras i ett begränsat antal bitar i minnet, vanligen i binär form:

| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

betyder $(01011100)_2 =$
$= 0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = (92)_{10}$

- Heltal
  Inga problem. Heltal upp till en viss storlek lagras exakt.
- Reella tal
  Reella tal kan inte lagras exakt utan måste avrundas eller huggas av. Representationen av reella tal i datorn kallas *flyttalsrepresentation* och talen kallas *flyttal*.

**Avrundningsfel och flyttalssystem**
Reella tal:
Reellt tal $x$ kan skrivas (exakt)

$$x = m \cdot \beta^e$$

där $\beta$ är den bas som används, $e$ exponent och mantissan

$$m = \pm(0.d_1d_2\ldots) = d_1\beta^{-1} + d_2\beta^{-2} + \ldots \quad , 0 \leq d_i < \beta$$

där

$$\beta^{-1} \leq |\, m \,| < 1 \quad \Rightarrow \; d_1 \neq 0 \text{ kallas normaliserad form}$$

Innebär alltså att decimalpunkten flyttas så att den första termen efter decimalpunkten, $d_1$ blir skild från noll.

**Avrundningsfel och flyttalssystem**

**Exempel** I bas $\beta = 10$:

$$\pi = 3.14159265\ldots = 0.314159265\ldots \cdot 10^1 =$$
$$(3 \cdot 10^{-1} + 1 \cdot 10^{-2} + 4 \cdot 10^{-3} + \ldots) \cdot 10^1.$$

Stämmer att $0 \leq d_i < 10$ och $0.1 \leq |\, m \,| < 1$.

När reella talet $x$ lagras i datorn görs det som flyttal, betecknas $fl(x)$. Hur ska reella talet representeras i datorns begränsade antal bitar? Exponenten $e$ kan bara lagras exakt upp till en viss storlek, och mantissan $m$ måste kapas eftersom den är oändlig.

**Avrundningsfel och flyttalssystem**

Ett flyttal $fl(x)$ representeras

$$fl(x) = \hat{m} \cdot \beta^e, \;\; \hat{m} = \pm(.d_1 d_2, \ldots, d_p), \;\; \text{där}$$
$$0 \leq d_i < \beta, \;\; d_1 \neq 0, \;\; L \leq e \leq U, \;\; (0 \text{ hanteras separat})$$

Detta medför att

$| \, fl(x) \, | \geq min$, annars "underflow"

$min = (1 \cdot \beta^{-1} + 0 \cdot \beta^{-2} + \ldots + 0 \cdot \beta^{-p})\beta^L = \beta^{L-1}$

$| \, fl(x) \, | \leq max$, annars "overflow".

$max = ((\beta - 1)\beta^{-1} + (\beta - 1)\beta^{-2} + \ldots + (\beta - 1)\beta^{-p})\beta^U =$

$(1 + \beta^{-1} + \beta^{-2} + \ldots + \beta^{-p-1} - \beta^{-1} - \ldots - \beta^{-p-1} - \beta^{-p})\beta^U =$

$(1 - \beta^{-p})\beta^U$,

**Avrundningsfel och flyttalssystem**

- Flyttalssystemet karaktäriseras alltså av $(\beta, p, L, U)$ på en given dator.
- $p$ kallas precision
- Vi antar att exponenten $e$ lagras exakt (eftersom heltal)
- Mantissan rundas av
- $\hat{m}$ och $e$ lagras. $\beta$ är fixt för en given dator och lagras ej. Vanligen gäller att $\beta = 2$.

**Avrundningsfel och flyttalssystem**

**Exempel)**

Flyttalssystemet $(\beta, p, L, U) = (2, 3, 0, 2)$.

$\hat{m}$ kan anta följande positiva värden (motsvarande för negativa tal):

$$(0.100) = \tfrac{4}{8} \quad (0.101) = \tfrac{5}{8} \quad (0.110) = \tfrac{6}{8} \quad (0.111) = \tfrac{7}{8}$$

och följdaktligen fås

| | | | | |
|---|---|---|---|---|
| $e = 0$ | $min = \frac{1}{2}$ | $\frac{5}{8} = 0.625$ | $\frac{6}{8} = 0.75$ | $\frac{7}{8} = 0.875$ |
| $e = 1$ | $\frac{2}{2} = 1$ | $\frac{10}{8} = 1.25$ | $\frac{12}{8} = 1.5$ | $\frac{14}{8} = 1.75$ |
| $e = 2$ | $\frac{16}{8} = 2$ | $\frac{20}{8} = 2.5$ | $\frac{24}{8} = 3$ | $max = \frac{28}{8} = 3.5$ |

OBS! Flyttalen är *ej jämnt fördelade*! Större tal – glesare representation.

**Avrundningsfel och flyttalssystem**

**Exempel, forts)**

Givet ett tal $x$ som ska representeras i datorn som flyttal. Hur stort kan felet maximalt bli?

Vid avrundning fås maximalt fel i mantissan då $m$ ligger mitt mellan två tal $\hat{m}$ i flyttalssystemet, t ex $m = \frac{11}{16}$ i exemplet. Antag avrundning uppåt. Vi får felet:

$$| \, \hat{m} - m \, | = | \, \tfrac{12}{16} - \tfrac{11}{16} \, | = \frac{1}{16}, \quad \text{dvs} \; | \, \hat{m} - m \, | \leq \frac{1}{16}$$

Allmänt gäller för felet i mantissan

$$| \, \hat{m} - m \, | \leq \frac{1}{2}\beta^{-p}$$

**Avrundningsfel och flyttalssystem**

För felet i hela talet, dvs $fl(x)$ gäller:

- Absoluta felet

$$| fl(x) - x | = | \hat{m}\beta^e - m\beta^e | = | (\hat{m} - m)\beta^e | \leq \frac{1}{2}\beta^{-p} \cdot \beta^e$$

  Felet beror av storleken på talet $x$, en följd av att flyttalen är glesare representerade för stora tal

- Relativa felet

$$\frac{| fl(x) - x |}{| x |} \leq \frac{\frac{1}{2}\beta^{-p} \cdot \beta^e}{| m | \cdot \beta^e} = \frac{1}{2}\frac{\beta^{-p}}{| m |} \leq \frac{1}{2}\frac{\beta^{-p}}{\beta^{-1}} = \frac{1}{2}\beta^{1-p}$$

  Beror *ej* på talets storlek !

**Avrundningsfel och flyttalssystem**

Vi har

$$\frac{| fl(x) - x |}{| x |} \leq \epsilon_M, \quad \text{där } \epsilon_M = \frac{1}{2}\beta^{1-p}$$

vid avrundning.

$\epsilon_M$ kallas *maskinepsilon* eller *avrundningsenheten* och är en maskinberoende konstant.

Det gäller att

$$\epsilon_M, \text{ är det minsta tal } \epsilon \text{ s.a } fl(1+\epsilon) > 1$$

I MATLAB: Kommandot » `eps` ger maskinepsilon

**Avrundningsfel och flyttalssystem**

**Exempel**

Antag $(\beta, p, L, U)$ och $\beta = 2$, dvs bas 2.

Vanligt antalet bitar i minnet, $p$ är

$p = 24 \Rightarrow \epsilon_M \approx 0.00000006 = 6 \cdot 10^{-8}$

$p = 53 \Rightarrow \epsilon_M \approx 2 \cdot 10^{-16}$

Normalt används 53 bitar, t ex av MATLAB. Detta ger som bäst ca 16 decimalers noggrannhet. Vid beräkningar av olika typ så gäller att alla relativa fel som är som är mindre än detta enbart är avrundningsfel!

Detta förklarar resultaten i slide 2!

**Avrundningsfel och flyttalssystem**

Vi kan nu skriva

$$\frac{| fl(x) - x |}{| x |} \leq \epsilon_M \Rightarrow \frac{fl(x)-x}{x} = \delta, \quad | \delta | \leq \epsilon_M$$

$$\Rightarrow \quad fl(x) = x + \delta x = x(1+\delta)$$

Motsvarande uttryck för felet vid beräkningar med de fyra räknesätten:

$$\frac{| fl(x \; op \; y) - (x \; op \; y) |}{| x \; op \; y |} \leq \epsilon_M$$

$$\Rightarrow \quad fl(x \; op \; y) = x(1+\delta_1), \quad | \delta_1 | \leq \epsilon_M$$

där *op* betecknar $+, -, \cdot, /$, dvs de fyra räknesätten.

**Avrundningsfel och flyttalssystem**

Avrundningsfelen vid beräkningar är vanligen mycket små i förhållande till andra fel (t ex i indata eller diskretisering).

Men, kan vara problem vid subtraktion $fl(x-y)$:

$$fl(fl(x)-fl(y)) = fl(x(1+\delta_1)-y(1+\delta_2)=$$
$$fl(x-y+x\delta_1-y\delta_2) = fl((x-y)(1+\tfrac{x}{x-y}\delta_1-\tfrac{y}{x-y}\delta_2)$$

där $\mid \delta_i \mid \leq \epsilon_M$.
Om $x-y$ litet relativt $x$ resp $y$ fås stort fel ! Detta kallas för *kancellation* och uppstår då två nästan lika stora tal subtraheras. Kan ofta undvikas genom omskrivning.
**Ex)** $\sqrt{1+x}-\sqrt{1-x} = \dfrac{2x}{\sqrt{1+x}+\sqrt{1-x}}$.

**Avrundningsfel och flyttalssystem**

Några konsekvenser:

- ej meningsfullt med tester av typen **if** x = y **then**
  istället **if** `abs(x-y) < tolerans` **then**
  eller **if** `abs(x-y) < tolerans · abs(x)` **then**
- undvik subtrahera nästan lika stora tal
- associativa och distributiva lagarna gäller ej flyttal
  Har i praktiken ingen betydelse.
- summera om möjligt termer i växande storleksordning
  Sällan något man bryr sig om – har liten betydelse.

**IEEE flyttalsrepresentation**

- under 60 och 70-talen hade varje dator tillverkare sitt eget flyttalssystem
- flyttalsstandard utvecklades under tidigt 80-tal och följdes av tillverkare som Intel o Motorola
- utvecklat av arbetsgrupp hos the Institute for Electrical and Electronics Engineers (IEEE)
- IEEE-standarden har tre viktiga krav
  - konsistent flyttalsrepresentation
  - korrekt avrundningsaritmetik
  - konsistent behandling av exceptionella situationer

**IEEE flyttalsrepresentation**

- tre standardtyper: **single precision**,**double precision** och **extended precision**
  - **Exempel)** IEEE enkel precision

    $$\boxed{\pm \quad e_1e_2e_3\ldots e_8 \quad d_1d_2\ldots d_{23}}$$

    tecknet 1 bit, exponenten 8 bitar, mantissan 23 bitar
  - **Exempel** IEEE dubbel precision

    $$\boxed{\pm \quad e_1e_2e_3\ldots e_{11} \quad d_1d_2\ldots d_{52}}$$

    tecknet 1 bit, exponenten 11 bitar, mantissan 52 bitar

**IEEE flyttalsrepresentation**

- Maskinepsilon blir

| | |
|---|---|
| IEEE single | $\epsilon_M = 2^{-23} \approx 1.2 \cdot 10^{-7}$ |
| IEEE double | $\epsilon_M = 2^{-52} \approx 1.1 \cdot 10^{-16}$ |
| IEEE extended | $\epsilon_M = 2^{-63} \approx 1.1 \cdot 10^{-19}$ |

- $(r, p, L, U)$ i IEEE-standard

| | |
|---|---|
| IEEE single | $(2, 24, -126, 128)$ |
| IEEE double | $(2, 53, -1022, 1024)$ |

Obs ”hidden bit”

**IEEE flyttalsrepresentation**

IEEE definierar 5 olika ”exceptions”:

- ”Invalid operation”, t ex $\frac{0}{0}$, $0 * \infty$, ges värdet NaN (Not a Number)
- division med 0, sätt till $\pm\infty$
- overflow, sätt till $\pm\infty$ eller största flyttal
- underflow, sätt till 0 (eller ”subnormalt” tal)
- korrekt avrundning av reellt tal (ej exceptionell situation egentligen)

**Diskretiseringsfel**

Förutom avrundningsfel finns även diskretiseringsfel. **Exempel**
Numerisk derivering med *differenskvot*:

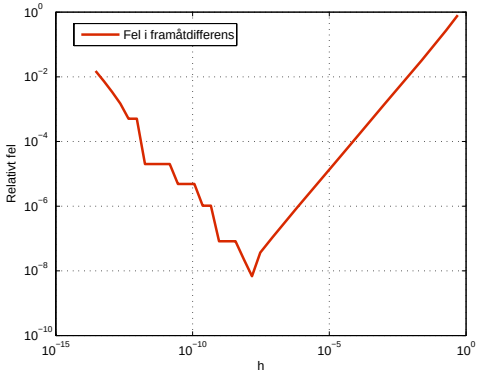
$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

När $h$ blir mindre borde approximationen bli successivt bättre eftersom finare indelning – mindre diskretiseringsfel.

Blir det så? Se resultat på labben.

**Diskretiseringsfel**

Hur blir det i praktiken?




Varför ser grafen ut som den gör?

**Diskretiseringsfel och avrundningsfel**
Tolkning:

- För stora $h$ spelar diskretiseringsfelet den dominerande rollen, kan bortse från avrundningsfelen
- För små $h$ är avrundningsfelet dominerande
- avrundningsfelet i det här fallet beror på
  - kancellation i täljaren för små $h$
  - division med litet tal förstärker felet i täljaren
- Avrundningsfelet uppför sig ”kaotiskt”, medan diskretiserkingsfelet ger en jämn och snygg kurva

**Diskretiseringsfel och avrundningsfel**
Sammanfattning:

- Diskretiseringsfelet spelar vanligen den dominerande rollen. Det är vanligt att man helt bortser från avrundningsfelen
- Avrundningsfel får konsekvenser i vissa fall, t ex kancellation. Om möjligt bör man ta hänsyn till detta i koden genom omskrivning.
- Exakt noll existerar i princip inte i praktiken! Om två reella tal $x$ och $y$ anses lika, skiljer de sig ändå i någon decimal.
- Ett relativt fel i storleksordningen $\epsilon_M$ efter beräkning med flyttal är enbart slumpmässigt ”skräp”.