Forelesning I

IMAT2150: Høsten 2023

Kent Holing, NTNU (22. august 2023)

22. og 23. august 2023 (forelesning 1 og 2)

Tema: Numeriske metoder. Generell numerikk: Representasjoner av flyttall i datamaskinen , beregninger med flyttall Pensum: Sauer 0.1-0.3.	Tema: Signifikans, Numeriske metoder. diverse feilkilder. Konvergensrate. Pensum: Sauer 0.4, 1.1-1.2

INNLEDNING

IMAT2150:
Litt om hva som etterstrebes i undervisningen og noen gode råd på veien

IMAT2150:
Litt om hva som etterstrebes i undervisningen og noen gode råd på veien

- Det foreleses på norsk.
- Det vil bli lagt stor vekt på å fortløpende motivere og øke forståelsen ved hjelp av gode eksempler.
- Læreboka inneholder mange og veldig gode eksempler og oppgaver. Dere oppmuntres til selv å bruk noe tid på i alle fall eksempler.

  (Det er dessverre ikke tid til å gå gjennom alle gode eksempler på forelesningene!)

IMAT2150:
Litt om hva som etterstrebes i undervisningen og noen gode råd på veien

- Det foreleses på norsk.
- Det vil bli lagt stor vekt på å fortløpende motivere og øke forståelsen ved hjelp av gode eksempler.
- Læreboka inneholder mange og veldig gode eksempler og oppgaver. Dere oppmuntres til selv å bruk noe tid på i alle fall eksempler.
  (Det er dessverre ikke tid til å gå gjennom alle gode eksempler på forelesningene!)
- Merk at det kreves at 80% av øvingene er godkjent for at en kan gå opp til eksamen!

IMAT2150:
Litt om hva som etterstrebes i undervisningen og noen gode råd på veien

- Det foreleses på norsk.
- Det vil bli lagt stor vekt på å fortløpende motivere og øke forståelsen ved hjelp av gode eksempler.
- Læreboka inneholder mange og veldig gode eksempler og oppgaver. Dere oppmuntres til selv å bruk noe tid på i alle fall eksempler.
  (Det er dessverre ikke tid til å gå gjennom alle gode eksempler på forelesningene!)
- Merk at det kreves at 80% av øvingene er godkjent for at en kan gå opp til eksamen!
- Forelesningsmaterialet vil fortløpende gjøres tilgjengelig rett etter forelesningene!

LÆRINGSMÅL

- Evaluering av polynom (bla. Horners metode) (S: 0.1)
- Binære tall - konvertering mellom desimaltall og binærtall (begge veier) (S: 0.2)
- Andre tallsystem (S: 0.2)
- Flyttall: Representasjon, formater og beregninger (S: 0.3)

Essentiell basiskunnskap

IMAT2150:
Læringsmål for forelesning I

1. Evaluering av polynom (bla. Horners metode)
2. Binære tall - konvertering mellom desimal tall og binære tall (begge veier)
3. Andre tallsystem
4. Reelle tall representert som flyttall (del 1)

Finne polynomverdier Naive metoder og Horner

Eksempel $P(x) = 3x^4 - 2x^3 - x^2 - 3x + 3$. Regn ut $P(x)$ for $x = 2$

Naiv metode Skriver ut alle potenser og regner ut
$P(2) = 3 \cdot 2 \cdot 2 \cdot 2 \cdot 2 - 2 \cdot 2 \cdot 2 \cdot 2 - 1 \cdot 2 \cdot 2 - 3 \cdot 2 + 3 = 25$
(10 multiplikasjoner 4 addisjoner.
Generelt: $\frac{d^2+3d}{2}$ regneoperasjoner.)

Litt mindre naiv metode Gjenbruker potenser for å regne ut neste potens $2^2 = 2 \cdot 2 = 4$, $2^3 = 2^2 \cdot 2 = 8$, $2^4 = 2^3 \cdot 2 = 16$
$P(2) = 3 \cdot 16 - 2 \cdot 8 - 1 \cdot 4 - 3 \cdot 2 + 3 = 25$
(7 multiplikasjoner 4 addisjoner.
Generelt: $3d - 1$ regneoperasjoner.)

Horners metode Kommer på neste side (4 multiplikasjoner 4 addisjoner.
Generelt: $2d$ regneoperasjoner.)

Horners metode

Minimerer antall utregner ved å nøste opp polynomet.

$$\begin{aligned} P(x) &= a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 \\ &= (a_4x^3 + a_3x^2 + a_2x + a_1)x + a_0 \\ &= ((a_4x^2 + a_3x + a_2)x + a_1)x + a_0 \\ &= (((a_4x + a_3)x + a_2)x + a_1)x + a_0 \end{aligned}$$

$P(2) = (((3 \cdot 2 - 2) \cdot 2 - 1) \cdot 2 - 3) \cdot 2 + 3 = 25$

2 d

(4 multiplikasjoner 4 addisjoner)

Polynomer med basepunkter er viktig i senere kapitler

$$Q(x) = (((a_4(x - b_4) + a_3)(x - b_3) + a_2)(x - b_2) + a_1)(x - b_1) + a_0$$

Hvorfor er dette så viktig?

EKSEMPEL

Gitt $p(x) = \mathbf{3x^4 - 2x^3 - x^2 - 3x + 3}$, bestem $p(2)$!

$$\mathbf{p(x) = (((3\,x - 2)\,x - 1)\,x - 3)\,x + 3}$$

EKSEMPEL

Gitt $p(x) = \mathbf{3x^4 - 2x^3 - x^2 - 3x + 3}$, bestem $p(2)$!

$$\mathbf{p(x) = (((3\,x - 2)\,x - 1)\,x - 3)\,x + 3}$$

$$3 \times 2 - 2 = 4$$
$$4 \times 2 - 1 = 7$$
$$7 \times 2 - 3 = 11$$
$$11 \times 2 + 3 = \mathbf{25}$$

Så, $p(2) = 25$ i overensstemmelse med tidligere beregning.

EKSEMPEL

Gitt $p(x) = \mathbf{3x^4 - 2x^3 - x^2 - 3x + 3}$, bestem $p(2)$!

$$\mathbf{p(x) = (((3\,x - 2)\,x - 1)\,x - 3)\,x + 3}$$

$$3 \times 2 - 2 = 4$$
$$4 \times 2 - 1 = 7$$
$$7 \times 2 - 3 = 11$$
$$11 \times 2 + 3 = \mathbf{25}$$

Så, $p(2) = 25$ i overensstemmelse med tidligere beregning.

```python
#Evaluate value of polynomial using Horner's method
def horner(poly,x):
    n=len(poly)
    result = poly[0]
    for i in range(1, n):
        result = result*x + poly[i]
    return result
```

IMAT2150:
Læringsmål for forelesning I

1. Evaluering av polynom (bla. Horners metode)
2. Binære tall - konvertering mellom desimal tall og binære tall (begge veier)
3. Andre tallsystem
4. Reelle tall representert som flyttall (del 1)

Først et tilbakeblikk på 10-tallsystemet som vi er vant til for å se på hvordan vi bestemmer hvert siffer i et heltall der:

Først et tilbakeblikk på 10-tallsystemet som vi er vant til for å se på hvordan vi bestemmer hvert siffer i et heltall der:

$$\begin{aligned} 1034 &= 103 \times 10 + 4 \\ 103 &= 10 \times 10 + 3 \\ 10 &= 1 \times 10 + 0 \\ 1 &= 0 \times 10 + 1 \end{aligned}$$

Vi får altså her sifrene i 1034 i motsatt rekkefølge.[1]

[1] Heltallsdivisjonen 1034/10 gir kvotienten 103 og resten 4 osv..

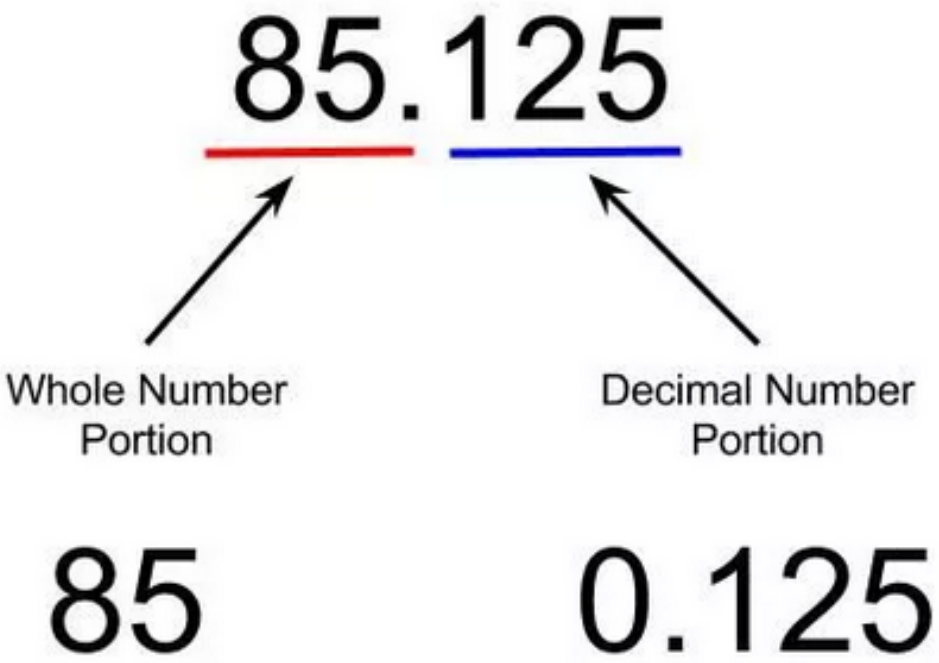
Konvertering av heltall til to-tall systemet skjer tilsvarende med heltalldivisjon med 2.

Vi skal også gi eksempler på hvordan vi konverterer desimaltall (brøker) til to-tallsystemet analogt til hva vi ville gjøre for å bestemme sifrene til brøker i 10-tallsystemet.

For å for eksempel bestemme desimalene i $n = 0.375$ bruker vi fortløpende multiplikasjon med 10:

$n \times 10 = 3.75$, $0.75 \times 10 = 7.5$, $0.5 \times 10 = 5$.

$$\ldots b_2 2^2 + b_1 2^1 + b_0 2^0 + b_{-1} 2^{-1} + b_{-2} 2^{-2} \ldots.$$



Whole Number Division	Result	Remainder
85/2	42	1
42/2	21	0
21/2	10	1
10/2	5	0
5/2	2	1
2/2	1	0
1/2	0	1

85 = 1010101

Når vi kommer til 0 i kolonne 2 ovenfor så stopper vi da det endrer ikke binær-representasjonen av heltallet om vi ikke gjør det.

Husk alltid å kontrollere utregningen![1]

[1] $(1010101)_2 = 1 + 4 + 16 + 64 = 85$ (OK!).

0.125 x 2	0.25	0
0.25 x 2	0.5	0
0.5 x 2	1.0	1
0.0 x 2	0.0	0

Altså (merk **endelig** binær-representasjon):[1]

$$\mathbf{85.125 = (1010101.001)_2}$$

[1] Det er lett å se at $(.001)_2 = 1/8 = 1/2^3 = 0.125$.

Merk at for $0.7 = 7/10 = 7/(2 \times 5)$ er nevneren ikke en toer-potens.

Fractional part. Convert $(0.7)_{10}$ to binary by reversing the preceding steps. Multiply by 2 successively and record the integer parts, moving away from the decimal point to the right.

$$
\begin{aligned}
.7 \times 2 &= .4 + 1 \\
.4 \times 2 &= .8 + 0 \\
.8 \times 2 &= .6 + 1 \\
.6 \times 2 &= .2 + 1 \\
.2 \times 2 &= .4 + 0 \\
.4 \times 2 &= .8 + 0 \\
&\vdots
\end{aligned}
$$

Notice that the process repeats after four steps and will repeat indefinitely exactly the same way. Therefore,

$$(0.7)_{10} = (.1011001100110\ldots)_2 = (.1\overline{0110})_2,$$

La oss kontrollere svaret!

Det nedenfor er en rettelse av det som ble vist på forelesningen!

La $x = (0.1\overline{0110})_2$.

Da er $2x = 1.\overline{0110}$.

Med $y = .\overline{0110}$ *er* $2^4\, y = 0110 + y$, så $y = 6/15 = 2/5$.

M.a.o. er $2x = 1 + y = 1 + 2/5 = 7/5$, og $x = 7/10 = 0.7$, som skulle vises.

Vi skal bestemme binær-representasjonen til 9.4, og vi skal bruke dette flere ganger senere for å illustrere nye begreper

Vi skal bestemme binær-representasjonen til 9.4, og vi skal bruke dette flere ganger senere for å illustrere nye begreper

Først, så bestemmer vi den for 0.4:

$$\begin{aligned} 2 \times 0.4 &= 0.8 + 0 \\ 2 \times 0.8 &= 0.6 + 1 \\ 2 \times 0.6 &= 0.2 + 1 \\ 2 \times 0.2 &= 0.4 + 0 \\ &\dots \end{aligned}$$

der dette (perioden) gjentar seg i det uendelige. (Vi sier at den binære representasjonen er periodisk.)

Vi skal bestemme binær-representasjonen til 9.4, og vi skal bruke dette flere ganger senere for å illustrere nye begreper

Først, så bestemmer vi den for 0.4:

$$\begin{aligned} 2 \times 0.4 &= 0.8 + 0 \\ 2 \times 0.8 &= 0.6 + 1 \\ 2 \times 0.6 &= 0.2 + 1 \\ 2 \times 0.2 &= 0.4 + 0 \\ &\cdots \end{aligned}$$

der dette (perioden) gjentar seg i det uendelige. (Vi sier at den binære representasjonen er periodisk.)

Da er $\mathbf{9.4} = (\mathbf{1001.0110} \cdots \mathbf{0110} \cdots \mathbf{0110} \cdots)_2$ siden $9 = 2^3 + 1 = (1001)_2$.[1] Vi skriver dette også som $9.4 = (1001.\overline{0110})_2$.

[1] Kontroll: Med $x = (.0110)_2$ har vi $2^4 x = 6 + x$, dvs. $x = 6/15 = 2/5 = 0.4$. (OK!) Merk at nevneren til x er forkortet mest mulig (2/5) ikke er en potens av 2.

1. Evaluering av polynom (bla. Horners metode)
2. Binære tall - konvertering mellom desimal tall og binære tall (begge veier)
3. Andre tallsystem
4. Reelle tall representert som flyttall (del 1)

Example 141.4:

$(141.4)_{10} = (10001101.\overline{0110})_2 = (8D.\bar{6})_{16} = (215.\overline{3146})_8$[1,2]

Det heksadesimale tallsystemet (16-talls-systemet):

Hex	Dec	Bin	Hex	Dec	Bin
0	0	0000	8	8	1000
1	1	0001	9	9	1001
2	2	0010	A	10	1010
3	3	0011	B	11	1011
4	4	0100	C	12	1100
5	5	0101	D	13	1101
6	6	0110	E	14	1110
7	7	0111	F	15	1111

Det oktadesimale tallsystemet (åtte-talls-systemet):

Hex	Dec	Bin	Hex	Dec	Bin
0	0	000	4	4	100
1	1	001	5	5	101
2	2	010	6	6	110
3	3	011	7	7	111

[1] Bestem selv $(141.4)_2$ der vi har $(0.4)_2$ fra tidligere.

[2] $(141.4)_8$ følger av at $(141.4)_8 = |010|001|101|.\overline{|011|001|100|110|}$.

Normaliserte reelle tall (såkalte flyttall) uttrykkes matematisk på formen

$$\pm 1.bbbbbbbbbbbb \cdots \times 2^e$$

($b = 0/1$; eksponent e),[1] men hvordan representeres de i datamaskinen?

[1] Som indikert tidligere, kan det her være uendelig mange bits.

Normaliserte reelle tall (såkalte flyttall) uttrykkes matematisk på formen

$$\pm 1.bbbbbbbbbbbb \cdots \times 2^e$$

($b = 0/1$; eksponent e),[1] men hvordan representeres de i datamaskinen?

Når du for eksempel taster inn et tall på terminalen og trykker enter, hva blir lagret internt?

[1] Som indikert tidligere, kan det her være uendelig mange bits.

Normaliserte reelle tall (såkalte flyttall) uttrykkes matematisk på formen

$$\pm 1.bbbbbbbbbbbb \cdots \times 2^e$$

($b = 0/1$; eksponent e),[1] men hvordan representeres de i datamaskinen?

Når du for eksempel taster inn et tall på terminalen og trykker enter, hva blir lagret internt?

Datamaskiner bruker (selvsagt) et endelig antall bits (0/1) for å lagre tall.

Som vi skal se, kan derfor regneoperasjoner gi oss resultater som både kan overraske og føre til store feil.

[1] Som indikert tidligere, kan det her være uendelig mange bits.

Normaliserte reelle tall (såkalte flyttall) uttrykkes matematisk på formen

$$\pm 1.bbbbbbbbbbbb \cdots \times 2^e$$

($b = 0/1$; eksponent e),[1] men hvordan representeres de i datamaskinen?

Når du for eksempel taster inn et tall på terminalen og trykker enter, hva blir lagret internt?

Datamaskiner bruker (selvsagt) et endelig antall bits (0/1) for å lagre tall.

Som vi skal se, kan derfor regneoperasjoner gi oss resultater som både kan overraske og føre til store feil.

Å være oppmerksom på, og kunne kontrollere dette, er en viktig del av fagfeltet numerisk analyse.

[1] Som indikert tidligere, kan det her være uendelig mange bits.

Binær-representasjon av reelle tall

Av eksemplene vi hittil har gitt på binær representasjon av reelle tall har vi sett at de kan være vesentlig forskjellig ettersom de er rasjonale eller irrasjonale tall.

Vi har:

SETNING

For r et reelt tall gjelder det at r er en brøk hvis og bare hvis r har enten en endelig eller en (uendelig) da periodisk binær-reprepresentasjon (b-repr.).

Et irrasjonalt tall har derfor en (uendelig) ikke-periodisk b-repr..

En brøk $r = a/b$ med $gcd(a, b) = 1$ (dvs. en mest mulig forkortet brøk)[1] har en endelig b-repr. hvis og bare hvis 2 er den eneste primtallsfaktoren til b eller $b = 1$ (for $a/b = a$ heltall).

[1] UTAG kan vi anta at a og b er positive heltall.

IMAT2150:
Læringsmål for forelesning I

1. Evaluering av polynom (bla. Horners metode)
2. Binære tall - konvertering mellom desimal tall og binære tall (begge veier)
3. Andre tallsystem
4. Reelle tall representert som flyttall (del 1)

Igjen, normaliserte reelle tall (såkalte flyttall) uttrykkes matematisk på formen $\pm 1.bbbbbbbbbbbb\cdots \times 2^e = \pm m \times 2^e$.

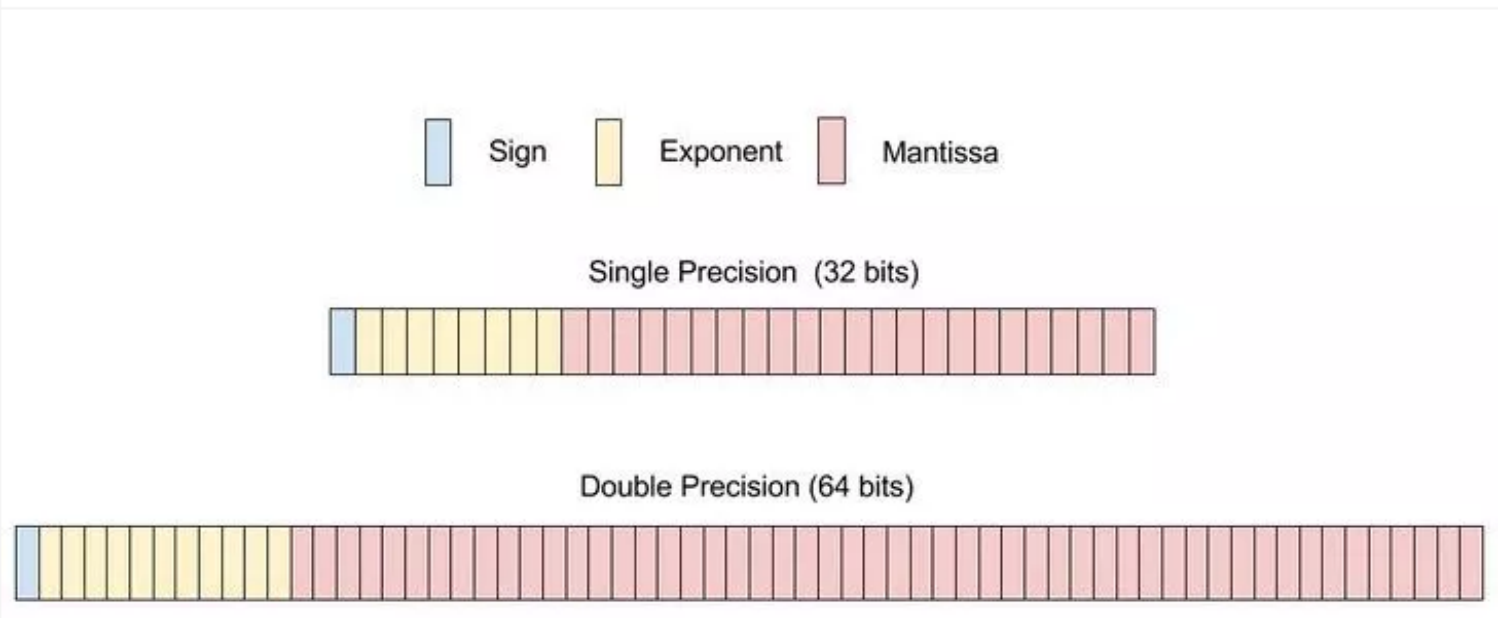
Her er $b = 0/1$; eksponenten e er heltallig og mantissen m oppfyller $1 \leq m < 2$.

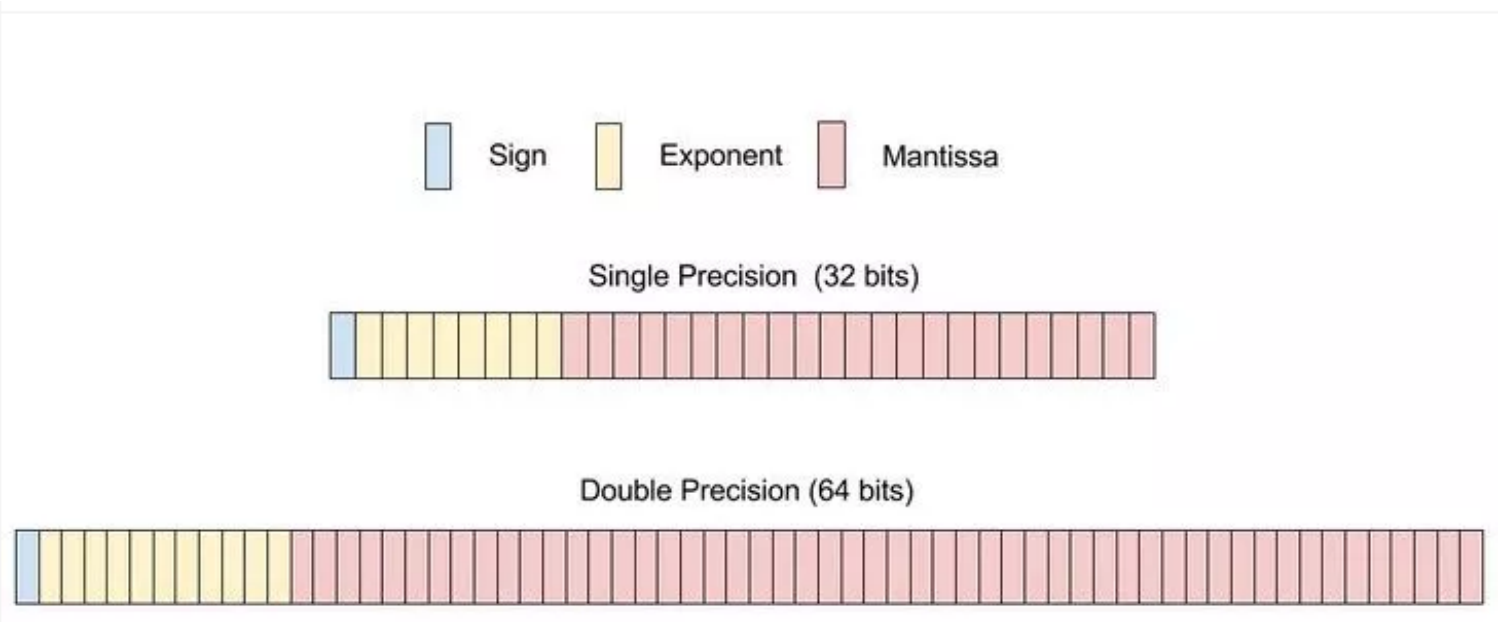
EKSEMPEL

$n = 23 = (10111)_2 = (1.0111)_2 \times 2^4$.

Her er altså $e = 4$ og $m = \frac{23}{16} = 1 + \frac{7}{16} = (1.4375)_{10} = (1.0111)_2$.[1]

[1] Her er $7/16 = 1/4 + 1/8 + 1/16$.


Sign
Exponent
Mantissa
Single Precision (32 bits)
Double Precision (64 bits)



SP: S(1), E(8) og M(23) / DP: S(1), E(11) og M(52)

Endelig representasjon av reelle tall på datamaskinen vil ofte føre til avrundingsfeil.
Hvis mange nok flyttallsoperasjoner utføres, så kan denne feiltypen akkumuleres og i verste fall ødelegge beregningen.

Vi skal konsentrere oss mest om DP, men bruker det enklere SP formatet som første eksempel.[1]

[1] Vi har også et tredje IEEE754 format dvs. `"longdouble"` med S(1), E(15) og M(64).

IMAT2150: IEEE754 Single Precision Format - Et første eksempel på IEEE754

Merk at det er en typo nedenfor: 19.25 skal være 19.59375.

Merk også at $19.59375 = 19 + 59375/100000 = 19 + 19/32$ og $19 = 1 + 2 + 16 = 1 + 2 + 2^4$ og $19/32 = 1/2 + 1/2^4 + 1/2^5$.[1]

Convert the denary value 19.25 into IEEE 754 standard 32 bit floating point binary

Sign bit	Exponent								Mantissa																							
0	1	0	0	0	0	0	1	1	0	0	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Step 1: Determine the sign bit (0 if positive, 1 if negative)

Sign bit = 0

Step 2: Convert to pure binary

	16	8	4	2	1	0.5	0.25	0.125	0.0625	0.03125
19.59375_{10} =	1	0	0	1	1	1	0	0	1	1

[1] Dette gir bitmønstrene nedenfor.

IMAT2150: IEEE754 Single Precision Format - Et første eksempel på IEEE754

Step 3: Normalise to determine the mantissa and the unbiased exponent *(place the binary point after leftmost 1)*

$1\ 0\ 0\ 1\ 1\ .\ 1\ \ 0\ \ 0\ \ 1\ \ 1\ = 1.001110011 \times 2^4$

Step 4: Determine the biased exponent *(add 127 then convert to an unsigned binary integer)*

$4 + 127 = 131_{10} = 10000011_2$

Step 5: Remove the leading 1 from the mantissa *(remove the leftmost 1)*

IMAT2150: IEEE754 Single Precision Format - Et første eksempel på IEEE754

Convert the denary value 19.25 into IEEE 754 standard 32 bit floating point binary

Sign bit	Exponent								Mantissa																							
0	1	0	0	0	0	0	1	1	0	0	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Step 1: Determine the sign bit (0 if positive, 1 if negative)

Sign bit = 0

Step 2: Convert to pure binary

	16	8	4	2	1	0.5	0.25	0.125	0.0625	0.03125
19.59375_{10} =	1	0	0	1	1 .	1	0	0	1	1

Step 3: Normalise to determine the mantissa and the unbiased exponent *(place the binary point after leftmost 1)*

1 0 0 1 1 . 1 0 0 1 1 = 1.001110011×2^4

Step 4: Determine the biased exponent *(add 127 then convert to an unsigned binary integer)*

$4 + 127 = 131_{10} = 10000011_2$

Step 5: Remove the leading 1 from the mantissa *(remove the leftmost 1)*

Maskin-representasjon (dobbel presisjon)

Flyttall med dobbel presisjon lagres med 64 bits i en datamaskin.

- **1** bit s til fortegnet.
  - $s = 0$ betyr positivt tall.
  - $s = 1$ betyr negativt tall.
- **11** bit til eksponenten e.
  - $-1022 \leq e \leq 1023$
  - Tallet $e + 1023$ lagres.
  - Eksponentbias $1023 = (011\ 1111\ 1111)_2$ trekkes fra for å få tilbake e
- **52** bit til mantissa $1 \leq m < 2$.
  - Heltallet $M = 2^{52}(m - 1)$ lagres.
  - Mantissa regnes ut med $m = 1 + M \cdot 2^{-52}$.

Tallet 1 i dobbel presisjon

Vi skriver $1 = +1 \times 2^0$.

- Derfor er $s = 0$, $e = 0$ og $m = 1$.
- Vi legger til eksponentbias $1023 = (011\ 1111\ 1111)_2$ til eksponenten e.
- Heltallet $M = 2^{52}(m-1) = 0$ lagres for mantissa.

Tallet 1 lagres derfor som

$$[0|011\ \ 1111\ \ 1111\ |\ 0000\ \ 0000\ \ 0000\ \ 0000\ \ 0000$$
$$0000\ \ 0000\ \ 0000\ \ 0000\ \ 0000\ \ 0000\ \ 0000\ \ 0000]$$

$$= \mathbf{3FF0\ \ 0000\ \ 0000\ \ 0000}$$

på hexadesimal-form.

Eksempel på binær-representasjon av et irrasjonalt tall

Eksempel: n $= \pi$ (π er et irrasjonalt tall)[1]

$\pi =$
$3.14159265358979323846264338327950288419716939937510\ldots =$
$(11.001001000011111101101010100010001000010110100011\cdots)_2$

Uendelig og ikke periodisk binær-representasjon.

Lagret i double precision IEEE754 format (64 bits) ekvivalent med

400921FB54442D18 (hex)

[1] Andre kjente irrasjonale tall er $\sqrt{2}$ og e.

Exponents

- 11 bits allow us to represent $2^{11} = 2048$ different integers
- We need both positive and negative exponents, and we keep two integers for special cases, so we have 2046 left to cover integers from -1022 to 1023:

binary	00000000000	00000000001	00000000010	$\cdots$	11111111110	11111111111
decimal	0	1	2	$\cdots$	2046	2047
exponent	special	-1022	-1021	$\cdots$	1023	special

- 1023 is the exponent bias. We add this amount to the actual exponent, and then store that number (note: this is the same as adding 1024 and subtracting 1)
- Subtract 1023 after storing (or subtract 1024 and add 1)

Vi kommer tilbake til de spesielle eksponentene.

Ofte er det ikke nok 52 bits i mantissa for å representere tallene eksakt. Hvordan avrunder vi til 52 bits når det er nødvendig?

Ofte er det ikke nok 52 bits i mantissa for å representere tallene eksakt. Hvordan avrunder vi til 52 bits når det er nødvendig?

Bitutdrag $\boxed{\cdots xxxXYyyy \cdots}$ (bit 52: X, bit 53: Y)

1) Hvis $Y = 0$, trunkér (avrund ned), dvs. behold X.

2) Hvis $Y = 1$, og ikke alle tilgjengelige y er 0, avrund opp ($X \to X + 1$).

3) Hvis $Y = 1$, **og alle tilgjengelige (kjente) y-er 0**, avrund opp hvis $X = 1$; avrund ned hvis $X = 0$, dvs. behold X.

Husk mentetall hvis en avrunder opp!

Ofte er det ikke nok 52 bits i mantissa for å representere tallene eksakt. Hvordan avrunder vi til 52 bits når det er nødvendig?

Bitutdrag $\boxed{\cdots xxxXYyyy \cdots}$ (bit 52: X, bit 53: Y)

1) Hvis $Y = 0$, trunkér (avrund ned), dvs. behold X.

2) Hvis $Y = 1$, og ikke alle tilgjengelige y er 0, avrund opp ($X \to X + 1$).

3) Hvis $Y = 1$, **og alle tilgjengelige (kjente) y-er 0**, avrund opp hvis $X = 1$; avrund ned hvis $X = 0$, dvs. behold X.

Husk mentetall hvis en avrunder opp!

Hovedregelen er altså at vi avrunder til nærmeste flyttall, men, regel 3) er et **unntakstilfelle** der vi avrunder slik at siste siffer som tas med er 0.[1]

[1] Merk at hvis vi benytter bare trunkering så vil det alltid bevege resultatene skjevt ("biased") mot 0.

Mer verbalt enn på forrige lysark kan vi formulere avrundings-reglene som:

HOVEDREGEL:
Et tall avrundes til nærmeste flyttall.

UNNTAK:
Når det er like stor avstand til de nærmeste flyttallene over og under tallet, avrundes det slik at siste siffer som tas med er 0, som vist nedenfor.

Følgende tall må rundes **opp**

$$1.[101010\ldots1001]\underline{1}000\ldots \times 2^{19} \approx 1.[101010\ldots1010] \times 2^{19} \quad \uparrow$$

Følgende tall må rundes **ned**

$$1.[1011\ldots1000]\underline{1}000\ldots \times 2^{3} \approx 1.[1011\ldots1000] \times 2^{3} \quad \downarrow$$

Analogt til 10-tallsystemet: 2,3650 ligger like langt over 2,36 som under 2,37; avrund til det jamne tallet, m.a.o. 2,36.

To spesielle IEEE754 DP flyttall

1) Merk at 2^{-53} er det største maskinrepresentative tallet mindre enn $\epsilon_M = 2^{-52}$ (`"machine epsilon"`), som oppfyller at flyttallet $1 + 2^{-53}$ er lik 1

2) Merk også at $1 + \epsilon_M = 1 + 2^{-52}$ er det minste flyttallet større enn 1

Eksempel https://www.binaryconvert.com

$fl(x)$ defineres til å være det nærmeste IEEE754 DP flyttallet til x.

Vis at for $x = 9.4$, så er $fl(x) - x = 2^{-49}/5$.[1]

Hint: Vis at $1 - (.\overline{1100})_2 = (.\overline{0011})_2 = 1/5$. (Vises på neste slide!)

[1] SAUER (0.8) s. 10 gir det ekvivalente 0.2×2^{-49}.

Eksempel https://www.binaryconvert.com

$fl(x)$ defineres til å være det nærmeste IEEE754 DP flyttallet til x.

Vis at for $x = 9.4$, så er $fl(x) - x = 2^{-49}/5$.[1]

Hint: Vis at $1 - (.\overline{1100})_2 = (.\overline{0011})_2 = 1/5$. (Vises på neste slide!)

Vi har fra tidligere at $9.4 = (1001.\overline{0110})_2$, så[2]

$$9.4 = 1.[0010\ 1100\ 1100 \cdots\ 1100]\overline{1100} \times 2^3 \qquad 2$$

$$fl(9.4) = 1.[0010\ 1100\ 1100 \cdots\ 1101]0000\ \times 2^3.$$

M.a.o. i følge hintet er da
$fl(x) - x = fl(9.4) - 9.4 = 2^{-52} \times (.\overline{0011})_2 \times 2^3 = 2^{-49}/5 \approx 3,5527 \times 10^{-16}$.

[1] SAUER (0.8) s. 10 gir det ekvivalente 0.2×2^{-49}.

[2] Bit nr. 52 (uthevet i rødt) for fl(9.4) har blitt avrundet opp.

IMAT2150: Bevis av hintet på forrige slide

Vis skal vise at (*) $1 - (.\overline{1100})_2 = (.\overline{0011})_2 = 1/5$.

La $x = (.\overline{0011})_2$.

Da er $2^4\, x = 11.(\overline{0011})_2$, så

$(2^4 - 1)x = (11)_2 = 3$, dvs. $15x = 3$ og derfor $x = 3/15 = 1/5$.

På samme måte viser vi at $(.\overline{1100})_2 = 4/5$, og derfor er (*) riktig.

Avrundingsfeil ved regning med flyttall

Vi definerer funksjonen fl som gir et flyttall i dobbel presisjon.

$$fl(x) = \text{ nærmeste flyttall til } x$$

Eksempel:

$$\begin{aligned} x_c = fl(9.4) &= 1.001011\ldots \text{ (40 utelatte sifre) } \ldots 001101\ 0000 \times 2^3 \\ x = 9.4 &= 1.001011\ldots \text{ (40 utelatte sifre) } \ldots 001100\ \overline{1100} \times 2^3 \end{aligned}$$

Absolutt feil = $|x_c - x| = 0.$ (52 nuller) $\overline{0011} \times 2^3 \approx 3.5527 \cdot 10^{-16}$.

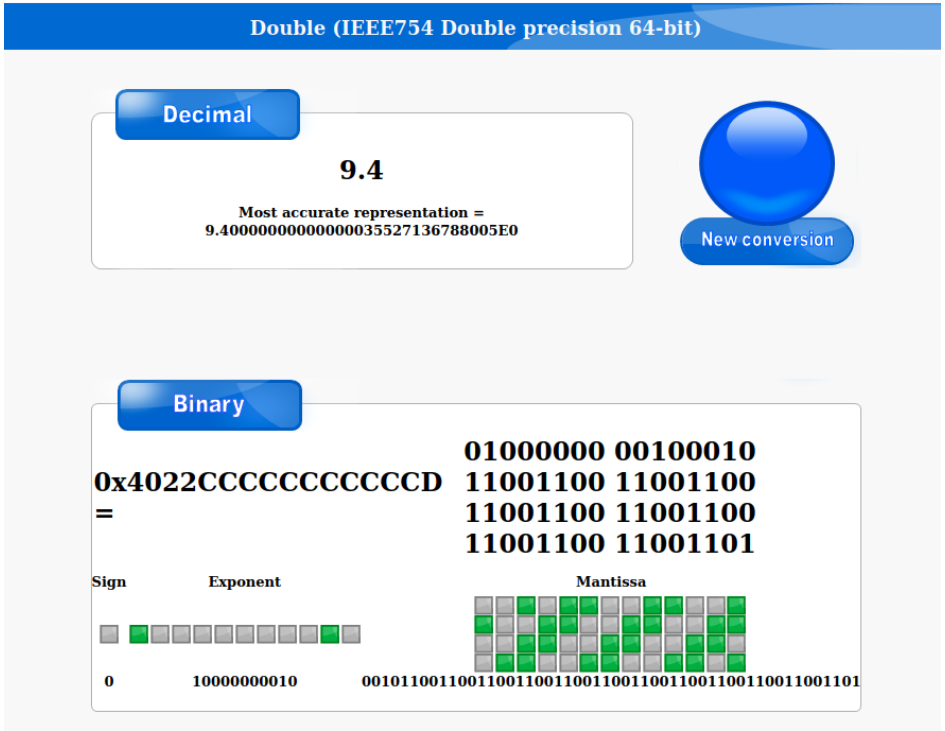
Relativ feil = $\frac{|x_c - x|}{|x|} \approx 3.7794 \times 10^{-17} \approx 0.17 \times \varepsilon_{mach}$.

Teorem (Setning om relativ feil til $fl(x)$)

$$\frac{|fl(x) - x|}{|x|} \leq \tfrac{1}{2}\varepsilon_{mach}.$$

IMAT2150: IEEE754 DP Online Converter

https://www.binaryconvert.com



IMAT2150:
Viser igjen en tidligere slide om IEEE754 DP eksponenter

Det er to (biased) eksponenter (0 og 2047) vi ennå ikke har omtalt (merket special), se nedenfor.

Exponents

- 11 bits allow us to represent $2^{11} = 2048$ different integers
- We need both positive and negative exponents, and we keep two integers for special cases, so we have 2046 left to cover integers from -1022 to 1023:

binary	00000000000	00000000001	00000000010	$\cdots$	11111111110	11111111111
decimal	0	1	2	$\cdots$	2046	2047
exponent	special	-1022	-1021	$\cdots$	1023	special

- 1023 is the exponent bias. We add this amount to the actual exponent, and then store that number (note: this is the same as adding 1024 and subtracting 1)
- Subtract 1023 after storing (or subtract 1024 and add 1)

Husk: I HEX er **7 = 0111 og F = 1111**.

7FF0 0000 0000 0000: **Inf** (eksempelvis 1/0) [$+\infty$]

(mantisse: bare 0-ere; biased eksponent består bare av 1-ere (dvs. er 2047); fortegnsbit 0)

FFF0 0000 0000 0000: **-Inf** (eksempelvis $-1/0$) [$-\infty$]

(mantisse: bare 0-ere; biased eksponent består bare av 1-ere (dvs. er 2047); fortegnsbit 1)

FFF* ** **** ****** : **NaN** (eksempelvis 0/0) ["not a number"]

(mantisse: ikke bare 0-ere)

Husk: I HEX er **8 = 1000**. **Små tall (såkalte subnormale tall)**

0000 0000 0000 0000 / **8000 0000 0000 0000**: ± 0[1]

[1] Her er **000** og **800** tilsammen fortegnsbiten og eksponenten ("biased") uttrykt i 16-tallssystemet. Mantissene er begge her lik 0.

Husk: I HEX er **8 = 1000**. **Små tall (såkalte subnormale tall)**

$\boxed{\mathbf{0000\ 0000\ 0000\ 0000}}$ / $\boxed{\mathbf{8000\ 0000\ 0000\ 0000}}$: $\pm\mathbf{0}$[1]

Når eksponenten er representert med bare 0-er så betyr maskinrepresentasjonen (her skrevet på binær på form!)

$\boxed{\mathrm{s00000000000b}_1 b_2 \cdots b_{52}}$ tallet $\pm\mathbf{0}.b_1 b_2 \cdots b_{52} \times 2^{-1022}$.

[1] Her er **000** og **800** tilsammen fortegnsbiten og eksponenten ("biased") uttrykt i 16-tallssystemet. Mantissene er begge her lik 0.

Husk: I HEX er **8 = 1000**. **Små tall (såkalte subnormale tall)**

0000 0000 0000 0000 / **8000 0000 0000 0000**: $\pm\mathbf{0}$[1]

Når eksponenten er representert med bare 0-er så betyr maskinrepresentasjonen (her skrevet på binær på form!)

s00000000000b$_1 b_2 \cdots b_{52}$ tallet $\pm\mathbf{0}.b_1 b_2 \cdots b_{52} \times 2^{-1022}$.

MERK: Her ikke den ledende biten 1, men 0!, som ellers er tilfelle for de normaliserte tallene vi hittil har diskutert.

0	00000000000	0000000000000000000000000000000000000000000000000001

Tallet ovenfor (dvs. $2^{-52} \times 2^{-1022} = 2^{-1074}$) er det minste ikke-null maskinrepresenterbare flyttallet ihht. IEEE754 DP formatet, dvs. reelle tall mindre enn 2^{-1074} er ikke representert i maskinen i det hele tatt.

[1] Her er **000** og **800** tilsammen fortegnsbiten og eksponenten ("biased") uttrykt i 16-tallssystemet. Mantissene er begge her lik 0.

ALGORITME

Floating point binary addition

- Make sure both numbers are normalised
- Make smaller exponent match the larger exponent
- Add mantissas together
- Normalise result if necessary

Machine addition consists of lining up the decimal points of the two numbers to be added, adding them, and then storing the result again as a floating point number. The addition itself can be done in higher precision (with more than 52 bits) since it takes place in a register dedicated just to that purpose. Following the addition, the result must be rounded back to 52 bits beyond the binary point for storage as a machine number.

For example, adding 1 to 2^{-53} would appear as follows:

$$
\begin{aligned}
& 1.\boxed{00\ldots0} \times 2^0 + 1.\boxed{00\ldots0} \times 2^{-53} \\
= \; & 1.\boxed{0000000000000000000000000000000000000000000000000000} \times 2^0 \\
+ \; & 0.\boxed{0000000000000000000000000000000000000000000000000000}1 \times 2^0 \\
\hline
= \; & 1.\boxed{0000000000000000000000000000000000000000000000000000}1 \times 2^0
\end{aligned}
$$

This is saved as $1. \times 2^0 = 1$, according to the rounding rule. Therefore, $1 + 2^{-53}$ is equal to 1 in double precision IEEE arithmetic. Note that 2^{-53} is the largest floating point number with this property; anything larger added to 1 would result in a sum greater than 1 under computer arithmetic.

Addering av flyttall er ikke en assosiativ operasjon:[1]

Prøv uttrykkene (1) $(x + y) + z$ og (2) $x + (y + z)$ for $x = 0.1$, $y = 0.3$ og $z = 0.00001$ i MATLAB eller Python

```
$ python
Python 2.7.17 (default, Mar  8 2023, 18:40:28)
[GCC 7.5.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> x=0.1
>>> y=0.3
>>> z=0.00001
>>> (x+y)+z
0.40001000000000003
>>> x+(y+z)
0.40001
```

[1] Det gjelder heller ikke for subtraksjon, multiplikasjon og divisjon. Men, merk at de elementære regneoperasjonene er kommutative.