Föreläsningsanteckningar i
Numerisk analys
MMG410 (MAN200, MAM240)
Thomas Ericsson, Matematik
Chalmers / GU
2009

Vad är numerisk analys?

Numerisk analys handlar om hur man löser beräkningsproblem på ett säkert och effektivt sätt med hjälp av dator.

Några viktiga komponenter:

- Problemets egenskaper
  - Problemen kommer från naturvetenskap, teknik, matematik etc.
  - Existerar det någon lösning?
  - Är den entydig?
  - Vad händer med lösningen när man ändrar indata något?
- Algoritmens egenskaper:
  - Hur snabb är metoden, implementationen?
  - Hur mycket minne går åt?
  - Vilka fel introduceras av algoritmen (avrundningsfel etc)?
- Beräkningsredskapets egenskaper:
  - Datorarkitektur. Möjligheter/begränsningar.
  - Parallellitet.
  - Programspråk och kompilatorer.

På följande sidor kommer några korta exempel på ovanstående, men först några ord om skillnaderna mellan ren matematik och numeriska beräkningar.

I grundläggande matematik-kurser ser man normalt endast problem som kan lösas exakt och med handräkningsmetoder. Det är därför enkelt att dra den felaktiga slutsatsen att alla problem kan lösas på detta vis.

Flertalet verkliga problem är dock för komplicerade att lösa exakt och även om det går så är det kanske inte intressant. Maple kan beräkna:

$$\int x^{10}e^x\,dx = \begin{array}{l}\left[3628800 - 3628800\,x + 1814400\,x^2 - 604800\,x^3 + \right.\\ \left. 151200\,x^4 - 30240\,x^5 + 5040\,x^6 - 720\,x^7 + 90\,x^8 - \right.\\ \left. 10\,x^9 + x^{10}\right]e^x + \text{konstant}\end{array}$$

och

$$\int_0^1 x^{10}e^x\,dx = 1334961\,e - 3628800$$

Jämför i Matlab

```
>> i = quadl(@(x) x.^10 .* exp(x), 0, 1)
i = 2.280015154878251e-01
```

med ett fel mindre än $4 \cdot 10^{-10}$. I en verklig tillämpning duger det kanske med fyra siffror.

$$\int_0^1 \arctan(e^x)\,dx$$

kan ej uttryckas på något enkelt sätt. Dock

```
>> i = quadl(@(x) atan(exp(x)), 0, 1)
i =
     1.017430583002258e+00
```

Dessa problem är enkla jämfört med många verkliga problem. Ett system av differentialekvationer kan normalt endast lösas approximativt och även det kan vara svårt (väderprognoser).

Hur många siffror behöver man i tillämpningar?
Några exempel:

Säg att vi skall tillverka en stålstav med längden $1m$. Hur många siffror är det rimligt att ange? De finaste passbitarna (dyra och mycket noggrant polerade mätblock) har en avvikelse på omkring $\pm 10^{-6}m$. En väteatom har en storlek $\approx 10^{-11}m$. Typen `double` i Java, C etc. tar 64 bitar, drygt 16 decimaler.

Vanliga elektriska motstånd har en osäkerhet om 5-10%, 0.1% för precisionsmotstånd.

Från NIST (National Institute of Standards and Technology) Special Publication 333, 2001 Edition, The International System of Units (SI) `http://physics.nist.gov/Pubs/pdf.html`

The unit of mass, the kilogram, is the mass of the international prototype of the kilogram kept at the BIPM (Bureau International des Poids et Mesures). It is a cylinder made of an alloy for which the mass fraction of platinum is 90 % and the mass fraction of iridium is 10 %. The masses of 1 kg secondary standards of the same alloy or of stainless steel are compared with the mass of the international prototype by means of balances with a relative uncertainty approaching 1 part in 10^9.

The mass of the international prototype increases by approximately 1 part in 10^9 per year due to the inevitable accumulation of contaminants on its surface. For this reason, the CIPM (Comité International des Poids et Mesures) declared that, pending further research, the reference mass of the international prototype is that immediately after cleaning and washing by a specified method (PV, 1989, 57, 104-105 and PV, 1990, 58, 95-97). The reference mass thus defined is used to calibrate national standards of platinum-iridium alloy (Metrologia, 1994, 31, 317-336).

In the case of stainless-steel 1 kg standards, the relative uncertainty of comparisons is limited to about 1 part in 10^8 by the uncertainty in the correction for air buoyancy. The results of comparisons made in vacuum, though unaffected by air buoyancy, are subject to additional corrections to account for changes in mass of the standards when cycled between vacuum and atmospheric pressure.

Mass standards representing multiples and submultiples of the kilogram can be calibrated by a conceptually simple procedure.

Slutsats: det räcker oftast med några få siffror.

Att förstå sitt problem

Ett generaliserat egenvärdesproblem: $Ax = \lambda Bx$.
A och B är stora och glesa matriser.

Saab: Ditt program gör fel. Vi får olika egenvärden varje gång.

$$A = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

Vi ser att

$$\begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 2 \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

men

$$\begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \lambda \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

för varje komplext tal λ. Singulärt matrisknippe.

Låt oss störa matriserna:

$$A = \begin{bmatrix} 2 & \epsilon \\ \epsilon & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & \delta \\ \delta & 0 \end{bmatrix}$$

Sekularekvationen, $\det(A - \lambda B) = 0$, blir:

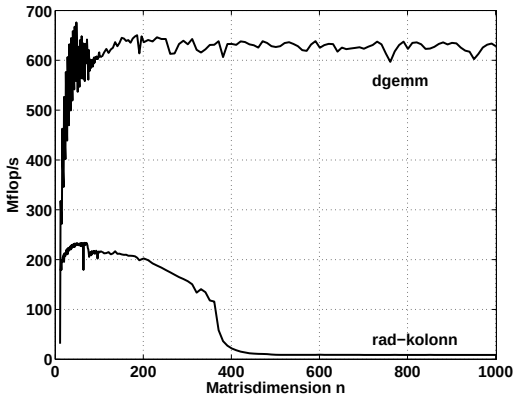
$$(\delta\lambda - \epsilon)^2 = 0$$

så att det dubbla egenvärdet är $\lambda = \epsilon/\delta$ för alla $\delta \neq 0$ och ϵ oavsett hur små de är.


```
>> A          % Ett linjärt ekvationssystem
A =
   -0.1537    0.8538    0.6535    0.1342
    0.6678   -0.1268   -0.1732   -0.1248
    1.5560   -1.0140   -1.0986   -0.5522
   -0.1248    0.0686    0.3664    0.3467

>> b
b =
    0.2042
   -0.1849
   -0.0358
   -0.9607

>> x = A \ b     % lös A x = b
x =
   -0.7033
    1.4745
   -1.4029
   -1.8333

>> f             % mätfel kanske
f =
   1.0e-10 *     % OBS: 1e-10
    0.3399
   -0.8218
    0.4035
    0.2154

>> A \ (b + f)
ans =
   1.0e+04 *     % OBS: 1e4
   -0.1257
   -2.1861
    3.5091
   -3.3217
```


Följande bild visar prestanda för två rutiner för matrismultiplikation, $A = BC$, där samtliga ingående matriser är kvadratiska med dimension n (x-axeln).

y-axeln visar antalet miljoner flyttalsoperationer (+, *) per sekund som flyttalsenheten (FPUn) i CPUn presterar. "dgemm" är hämtad från IBMs beräkningsbibliotek, ESSL (Engineering Scientific Subroutine Library). Rutinen ligger nära maskinens teoretiska toppfart.
"rad-kolonn" är algoritmen från kursen i linjär algebra. Det är inget fel på denna metod när man handräknar, men den är hopplöst ineffektiv på moderna datorer. Det beror på att FPUn är mycket snabbare än primärminnet: FPUn får vänta på matriselement att räkna med. I dgemm-rutinen har accessmönstret av matriselementen ändrats så att cache-minnena kan utnyttjas mer effektivt, vilket leder till inga eller korta väntetider.
Kodning blir mer komplicerad, vilket är en av flera anledningar till att vi inte ska studera verkliga implementationer.



Mål med kursen: numerisk allmänbildning.

Kunna svara på frågor som (för några problemklasser):

- Hur fungerar numeriska algoritmer?
- Hur ser typisk numerisk programvara ut?
- Vilka typer av problem, inom problemklassen, är möjliga att lösa?
- Är problemet svårt eller enkelt?
- Vilken programvara kan jag välja bland?
- Kan jag lita på resultatet?
- Tog beräkningen rimlig tid?
- Gick det åt för mycket minne?
- Hur stort problem av denna typ kan jag lösa?

Målet är inte att Du skall kunna skriva numerisk programvara. Vi kommer att studera principer, inte verkliga implementationer eftersom dessa är alltför tekniskt komplicerade.

Kursinnehåll

1. Konditionstal, stabilitet
2. Flyttalsaritmetik
3. $Ax = b$
4. Minstakvadratproblem
5. System av ickelinjära ekvationer
6. Interpolation
7. Kvadratur
8. Ordinära differentialekvationer
9. HPC (High Performance Computing)

Laborationer:

1. Flyttalsaritmetik, $Ax = b$
2. Minstakvadratproblem samt $f(x) = 0$
3. Kvadratur och ODE

Diverse felkällor

Fel som vi som numeriker inte kan göra så mycket åt

- modellfel, bortser från luftmotstånd, friktion
- mätfel, vågar etc. är inte exakta mellanavrundningar

Vi är intresserade av olika typer av beräkningsfel:

Avrundningsfel:

```
>> 49 * (1 / 49) - 1
ans = -1.1102e-16
```

Trunkeringsfel : $e^x \approx \sum_{k=0}^{N} \frac{x^k}{k!}$

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

Viktigt att välja "lagom stort" h.

Om stort och smått

Låt $\hat{x}$ vara en approximation av det exakta värdet x.

- absoluta felet $= \hat{x} - x$
- relativa felet $= \frac{\hat{x}-x}{x}$, om $x \neq 0$

Absoluta fel är ointressanta om vi inte vet ungefär hur stort x är.

Är 1.4 ett stort absolut fel? Ja, om det exakta värdet är 2, men inte om det exakta värdet är 10^9.
De relativa felen är 0.7 respektive $1.4 \cdot 10^{-9}$.

På samma sätt kan det absoluta felet 10^{-20} vara stort eller litet. Det är viktigt att känna till problemets skalning.
Vilken storleksordning har de tal vi arbetar med?

Relativa fel säger något även om vi inte känner till problemets skalning. Vi kommer därför att vara mer intresserade av relativa fel än av absoluta fel.

Tema: nollställen till polynom

Beräkna rötterna till $(x-1)^5 = 0$ i Matlab (där vi räknar med 16 siffror). Matlab vill ha en vektor med koefficienter:

$$(x-1)^5 = x^5 - 5\,x^4 + 10\,x^3 - 10\,x^2 + 5\,x - 1$$

```
>> r = roots([1 -5 10 -10 5 -1]) % koefficienter

r = 1.0008 + 0.0006i              % rötterna
    1.0008 - 0.0006i
    0.9997 + 0.0009i
    0.9997 - 0.0009i
    0.9990

>> abs(r - 1)                     % felen?
ans = 1.0e-03 *
       0.9625
       0.9625
       0.9625
       0.9625
       0.9625
```

Varför? Lös

$$(x-1)^5 = \epsilon \quad \Rightarrow \quad x = 1 + \epsilon^{1/5}$$

Om $\epsilon = 10^{-15}$ så är $\epsilon^{1/5} = 10^{-3}$. Nollställena till polynomet $(x-1)^5$ är tydligen känsliga för störningar i koefficienterna.

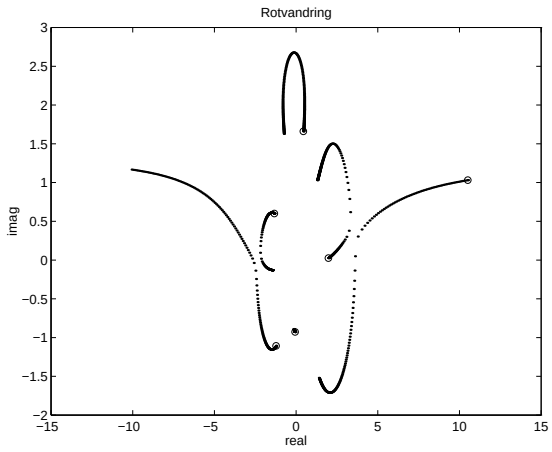
Är det alltid svårt att beräkna nollställen?

```
>> c = [1 -15 85 -225 274 -120];  % koefficienter
>> r = roots(c);  % de exakta röttena = 1, 2, 3, 4, 5
>> fel = sort(r) - (1:5)'
fel =
  -4.9960e-15
   6.6613e-14
  -1.5010e-13
   9.6811e-14
  -8.8818e-16
```

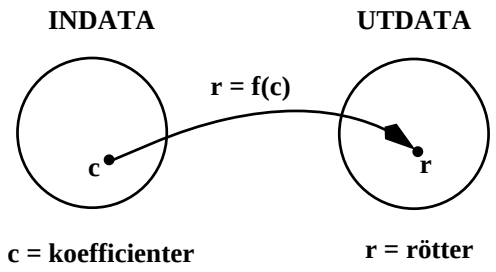
Rötter hos ett komplext 6-egradspolynom när vi stör x^5-termen.

$$x^6 + (c_5 + k\ \delta)x^5 + \cdots + c_0, \quad k = 0, 1, 2, 3 \ldots$$

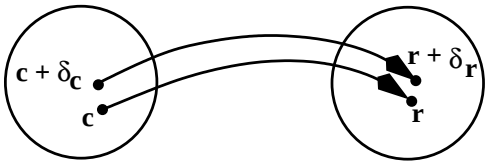
Ringarna visar rötternas begynnelsepositioner, $k = 0$.



Vi kan betrakta rötterna som funktioner av koefficienterna.



Vad händer när vi stör koefficienterna (indata i det allmänna fallet)? Det är viktigt att känna till om vårt problem är känsligt eller ej.



Om liten relativ ändring av indata ger en liten relativ ändring av resultatet säger man att det aktuella problemet är välkonditionerat. Om resultatet ändrar sig mycket är problemet illakonditionerat.

Konditionstalet är kvoten mellan de relativa förändringarna, dvs.

$$\boxed{konditionstalet = \frac{|\delta_r|/|r|}{|\delta_c|/|c|}}$$

Att beräkna konditionstalet är inte alltid möjligt; det kan vara lika svårt som att lösa det egentliga problemet. För vissa problemtyper är det överkomligt. Ibland är det dock möjligt att konstruera en uppskattning κ så att

$$\frac{|\delta_r|}{|r|} \leq \kappa \frac{|\delta_c|}{|c|}$$

Det räcker att känna till storleksordning på κ.
Är $\kappa \approx 10$ eller är $\kappa \approx 10^8$?

Exempel:
Hur känsliga är rötterna, till ekvationen $x^2 + ax + b = 0$, för ändringar i a och b?

Rötterna r_1 och r_2 är funktioner av a och b; $r_1(a,b)$, $r_2(a,b)$.

Låt r beteckna en av rötterna och låt $r + \delta_r$ beteckna den störda roten när vi ändrar koefficienterna med δ_a respektive δ_b. Vi har sambandet:

$$(r + \delta_r)^2 + (a + \delta_a)(r + \delta_r) + b + \delta_b = 0$$

Detta kan skrivas

$$\underbrace{r^2 + ar + b}_{0} + \delta_r(2r + a) + \delta_a r + \delta_b + \underbrace{\delta_r^2 + \delta_a\delta_r}_{\approx 0} = 0$$

Vi får det approximativa sambandet:

$$\delta_r \approx -\frac{\delta_a r + \delta_b}{2r + a} \quad \Rightarrow \quad |\delta_r| \lessapprox \frac{|\delta_a r| + |\delta_b|}{|2r + a|}$$

Eftersom r_1 och r_2 är rötter så gäller att:

$$\underbrace{(x - r_1)(x - r_2)}_{x^2-(r_1+r_2)x+r_1r_2} \equiv x^2 + ax + b \;\Rightarrow\; -(r_1 + r_2) = a$$

alltså är $2r_1 + a = r_1 - r_2$, gapet. Låt $g = |r_1 - r_2|$, då gäller:

$$|\delta_r| \lessapprox \frac{|\delta_a r| + |\delta_b|}{g}$$

Slutligen vill vi ha relativa störningar:

$$\frac{|\delta_r|}{|r|} \lessapprox \frac{1}{|r|}\left[\frac{|r||a|}{g}\frac{|\delta_a|}{|a|} + \frac{|b|}{g}\frac{|\delta_b|}{|b|}\right] \leq \underbrace{\frac{|a| + |b/r|}{g}}_{\approx\ konditionstalet} \max\left[\frac{|\delta_a|}{|a|}, \frac{|\delta_b|}{|b|}\right]$$

Observera att detta är en uppskattning av konditionstalet. Det är inte heller beräkningsbart eftersom vi måste känna r_1 och r_2. I praktiken kan man (kanske) uppskatta r_1 och r_2 med de beräknade rötterna. En viktig lärdom är att vi nu vet att gapet mellan rötterna är viktigt.

Bakåtanalys

Vi har sett s.k. framåtanalys: givet δ_c vad blir

$$f(c + \delta_c) - f(c)$$

Detta kan, som vi har sett, ge väldigt pessimistiska svar.

Ett alternativ är följande: givet approximationen $\hat{r}$ till det exakta värdet r hur mycket måste vi ändra c för att $\hat{r}$ skall bli en exakt lösning till det störda problemet? Vi söker alltså δ_c sådant att

$$f(c + \delta_c) = \hat{r}$$

Man kallar detta bakåtanalys. Detta på grund av att vi tittar på indatasidan i stället för på resultatsidan.

Exempel: Låt

$$p(x) = (x - 1)(x - 1.0001) = x^2 - 2.0001\ x + 1.0001$$

Vi vet sedan tidigare att konditionstalet har storleksordningen $1/(1.0001 - 1) = 10^4$.

Antag att vi på något sätt har producerat de dåliga approximativa rötterna 1.11 och 0.895. De relativa felen är ungefär 11%.

Det störda polynomet (som har rötterna 1.11 och 0.895) är:

$$(x-1.11)(x-0.895) = x^2-2.005x+0.99345 \;\Rightarrow\; \begin{cases} |\delta_a| \leq 5 \cdot 10^{-3} \\ |\delta_b| \leq 7 \cdot 10^{-3} \end{cases}$$

Detta innebär att vi har löst "nästan rätt problem"; vi har gjort ett relativt bra jobb med att beräkna rötterna. Att våra rötter är dåliga approximationer beror på att problemet är illakonditionerat.

Stabila algoritmer

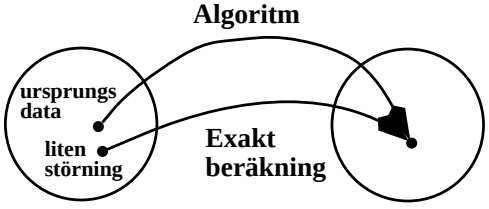
Man kan ofta lösa ett beräkningsproblem på olika sätt.

$$x^2 + ax + b = 0 \quad \text{har rötterna} \quad -\frac{a}{2} \pm \sqrt{\left(\frac{a}{2}\right)^2 - b}$$

Får vi ett bra program om vi skriver in ovanstående formler i koden? Nej, inte alltid. I en dator räknar vi med begränsat antal siffror. Om a är mycket större än b så kanske inte b kommer med när vi beräknar $(\frac{a}{2})^2 - b$. Detta svarar mot att $b = 0$ så att en beräknad rot blir noll. I detta fall är gapet stort så att rötterna är välkonditionerade.

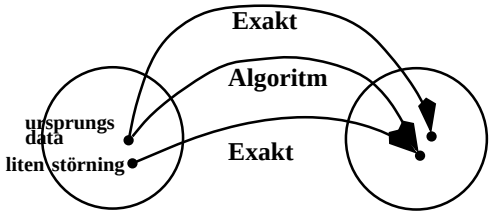
Det är algoritmen det är fel på! Det är i allmänhet ingen bra idé att kopiera formler direkt ur böcker.
Det finns bättre algoritmer; se övningarna.

En stabil algoritm genererar resultat som är exakta för ett lite stört problem.

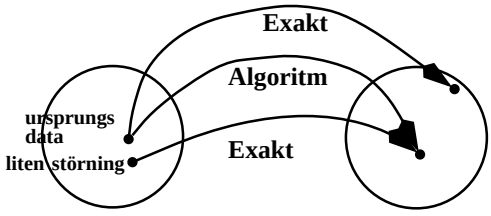


Om vi löser ett problem med en stabil algoritm får vi då ett resultat med ett litet fel?
Det beror på om det lilla bakåtfelet förstoras eller ej.

> Välkonditionerat problem + stabil algoritm ⇒ litet fel i resultatet.



Om vi applicerar en stabil algoritm på ett illakonditionerat problem löser algoritmen fortfarande nästan rätt problem.
Det lilla felet i indata kan dock ge upphov till ett stort fel i resultatet.



Flyttalsaritmetik

Flyttal (tal med flytande decimalpunkt):

$$x = \pm \left(d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \ldots + \frac{d_{t-1}}{\beta^{t-1}} \right) \beta^e, \;\; 0 \le d_k \le \beta - 1, \;\; L \le e \le U$$

β bas (vi kommer att ha $\beta = 2$)
t precision
$[L, U]$ exponentomfång
$d_0 d_1 \cdots d_{t-1}$ mantissa

IEEE 754 (Institute of Electrical and Electronics Engineers, Inc.) definierar enkel och dubbel precision (bland annat).
Vi antar att $\beta = 2$ från och med nu:

bas	t	L	U	
2	24	-126	127	32 bitar
2	53	-1022	1023	64 bitar

Ett tal $\neq 0$ är normaliserat om $d_0 \neq 0$.
Om $\beta = 2$ så är då $d_0 = 1$ varför man inte lagrar d_0.

I minnet lagras ett 64-bitars flyttal med en teckenbit, en exponent till vilken man adderat 1023 (för att slippa tecken på exponenten bl.a), samt mantissan (utom den inledande ettan).

Exempel: -3.25 är $(-1)^1 \cdot 1.101 \cdot 2^1$ i binär form.
Teckenbiten är ett, exponenten (ett) lagras som $1 + 1023 = 2^{10}$ och av mantissan lagrar vi 101, varför vi bör hitta följande bitar i minnet:

```
1  100 0000 0000         1010 0000 ... 0000 0000
s  exponent 11 bitar     mantissa 52 bitar lagras
```

Hexadecimalt (bas 16) gruppera fyra bitar / hex-siffra

```
c00a000000000000  (a <-> 1010, c <-> 1100)
```

a=10, b=11, c=12, d=13, e=14 och f=15.

I Matlab

```
>> format hex
>> -3.25
ans = c00a000000000000
```

Det finns speciella bitformat för diverse specialfall, t.ex.:

```
>> [0, -0]
ans = 0000000000000000   8000000000000000
```

Antalet olika tal är: $2(\beta - 1)\beta^{t-1}(U - L + 1) + 1$
dvs $4.2614 \cdot 10^9$ i enkel precision och $1.8429 \cdot 10^{19}$ i dubbel.
Att testa en funktion för alla flyttal i dubbel precision är nästan omöjligt. 10^9 tester per sekund ger 584.4 år.

Minsta positiva representerbara normaliserade talet $= 2^L \approx 1.17 \cdot 10^{-38}$ i enkel och $\approx 2.2 \cdot 10^{-308}$ i dubbel.

Det största representerbara talet har största exponenten och ettor i hela mantissan. I enkel precision $\approx 3.4 \cdot 10^{38}$, i dubbel $1.8 \cdot 10^{308}$. Tal större än denna gräns ger overflow.

```
>> 1e-200^2        % e har INGET med exp att göra
ans = 0            % underflow

>> 1e200^2         % overflow
ans = Inf          % Infinity

>> log(0)
ans = -Inf         % -Infinity

>> sin(1/0)
ans = NaN          % Not a Number
```


Ett enkelt talsystem med $\beta = 2, t = 4, L = -3, U = 3$.

Samma system inzoomat kring nollan.

Att lägga märke till: ett minsta positivt tal, ett största tal, ej kontinuerligt, grupperat, gap kring nollan, ökande avstånd mellan talen.

Vad är avståndet, gapet, mellan två grannar?

Det absoluta gapet ökar med talets storlek.
I dubbel precision är det absoluta gapet $\approx 2.2 \cdot 10^{-16}$ kring ettan, men $\approx 2 \cdot 10^{292}$ kring overflowgränsen.

Den relativa skillnaden, gapet dividerat med talet, varierar något med talet, men är ungefär $1.1 \cdot 10^{-16}$ till $2.2 \cdot 10^{-16}$ i dubbel precision.

Vi räknar med ungefär 16 decimala decimaler i dubbel precision.

Exempel:
för vilka $k \ge 0$ kan 10^k lagras exakt i dubbel precision?

10^{22} kan representeras exakt.
10^{23} lagras som 99999999999999991611392.

Det absoluta felet verkar "stort":

$99999999999999991611392 - 10^{23} = -8388608$.

Det är nu inte så farligt som det verkar, eftersom det relativa felet är:

$8388608/10^{23} \approx 8.3 \cdot 10^{-17}$.

Negativa exponenter är "svårare", 0.1 kan t.ex. inte lagras exakt i binärt format med ändlig mantissa.

$$\sum_{k=1}^{\infty} \left[\frac{1}{2^{4k}} + \frac{1}{2^{4k+1}} \right] = \frac{3}{2} \sum_{k=1}^{\infty} \frac{1}{2^{4k}} = \frac{3}{2} \sum_{k=1}^{\infty} \frac{1}{16^k} = \frac{3}{2} \, \frac{1}{16} \, \frac{1}{1 - 1/16} = 0.1$$

varför den binära framställningen av 0.1 kan skrivas;

$$0.0001100110011001100110011\ldots$$

Detta är inget konstigt. $1/3 = 0.3333\cdots$ i basen 10 men 0.1 i basen 3.

Om x är ett godtyckligt reellt tal betecknar vi det avrundade flyttalet med $fl(x)$ (floating). Normalt (kan ändras) är $fl(x)$ det flyttal som ligger närmast x.

Exempel: Låt oss anta att vi räknar decimalt med fyra siffror.
$fl(\pi) = fl(3.141592653589...) = 3.142$.
$fl(31415926.53589...) = 3.142 \cdot 10^7$.

Hur stort kan det absoluta felet bli vid avrundning till närmaste flyttal? Maximalt en halv enhet i fjärde siffran. Så om vårt tal är $\pm s_1.s_2s_3s_4 \ldots \cdot 10^e$ (där $s_1, s_2, \ldots$ betecknar decimala siffror) är absolutbeloppet av absoluta felet maximalt $0.0005 \cdot 10^e$.

Relativa felet är maximalt (för ett normaliserat tal)

$$\left|\frac{fl(x)-x}{x}\right| \leq \frac{0.0005 \cdot 10^e}{1.0000\ldots \cdot 10^e} = 0.0005$$

Denna begränsning kallas relativa maskinnoggrannheten och betecknas med ϵ_{mach}. Denna kvantitet bestäms av hur många siffror vi har i mantissan. Många siffror ger ett litet ϵ_{mach}.

Observera att ϵ_{mach} INTE HAR NÅGOT med exponentomfånget att göra (hur stora eller små tal vi kan arbeta med). Exponentomfånget ges ju av parametrarna L, U.

Om vi antar att vi räknar decimalt och $U = 100$ är det största talet vi kan lagra $9.999 \cdot 10^{100}$.

Om $L = -100$ är det minsta positiva representerbara talet $1.000 \cdot 10^{-100}$. Det minsta representerbara talet är INTE 0.0005.

I dubbel gäller att $\epsilon_{mach} = \frac{1}{2^t} \approx 1.11 \cdot 10^{-16}$ (16 siffrorna!) och i enkel precision $\epsilon_{mach} \approx 6 \cdot 10^{-8}$.

Vi kan skriva

$$\left|\frac{fl(x)-x}{x}\right| \leq \epsilon_{mach}$$

på ett annat sätt. Det gäller att:

$$fl(x) = (1+\epsilon)x = x + \epsilon x \quad \text{med} \quad |\epsilon| \leq \epsilon_{mach}$$

OBS OLIKA ϵ.

Varför gäller detta? Antag $x \neq 0$.

$$fl(x) = (1+\epsilon)x \Leftrightarrow fl(x) - x = \epsilon x \Rightarrow \underbrace{\left|\frac{fl(x)-x}{x}\right|}_{\leq \epsilon_{mach}} = |\epsilon|$$

OK även om $x = 0$ ty $fl(0) = 0$.

Enligt IEEE skall en enstaka `+, -, *, /` avrundas korrekt. Låt $\otimes$ beteckna någon av dessa operationer och låt x och y vara två flyttal. Då gäller att:

$$fl(x \otimes y) = (1+\epsilon)(x \otimes y) = x \otimes y + \epsilon(x \otimes y), \quad |\epsilon| \leq \epsilon_{mach}$$

förutsatt att vi inte får `Inf` eller `NaN`.

Beloppet av absoluta felet vi denna beräkning är alltså:

$$|fl(x \otimes y) - (x \otimes y)| = |\epsilon(x \otimes y)| \leq \epsilon_{mach}|x \otimes y|$$

och det relativa felet (om $x \otimes y \neq 0$):

$$\left|\frac{fl(x \otimes y) - (x \otimes y)}{x \otimes y}\right| = |\epsilon| \leq \epsilon_{mach}$$

Några vanliga problem med flyttalsräkning:

Utskiftning

Antag att vi räknar i dubbel precision:

```
>> a = 1e16;
>> b = a;                    % spara
>> a = a + 1
a = 1.000000000000000e+16

>> a - b
ans = 0
```

ger ingen förändring, ettan "trillar över kanten".

```
>> a = 1e16;
>> a = a + 123
a = 1.000000000000012e+16
>> a - b
ans = 124
```

Inte hela 123 kommer med.

Man bör undvika att addera eller subtrahera tal av mycket olika storleksordning.

$a + (b + c)$ behöver inte vara lika med $(a + b) + c$.
En kompilator får inte optimera för mycket.

```
>> 1 + (1e16 + (-1e16))
ans = 1

>> (1 + 1e16) + (-1e16)
ans = 0
```


Kancellation - subtraktion av två nästan lika stora tal

Antag att vi subtraherar följande två tal:

```
1.03678947f  f betecknar ett fel, en osäker siffra
1.03678935g  g betecknar ett fel, en osäker siffra
-----------
0.00000012t  t nytt fel
```

I de två första talen kommer felet i tionde siffran. I skillnaden finns felet redan i tredje siffran. Vi har alltså mycket större relativ osäkerhet i skillnaden än i någon av de ingående termerna. Vi känner alltså termerna mycket bättre än skillnaden; vi har förlorat information.
Vi får denna osäkerhet även om vi utför subtraktionen exakt.

Ett fl-exempel

Antag att a, b och c är (redan avrundade) flyttal och att vi vill beräkna $a + bc$. Vi får införa en ϵ-term för varje räkneoperation:

$$fl(a+bc) = fl(a+fl(bc)) = fl(a+bc(1+\epsilon_1)) = (a+bc(1+\epsilon_1))(1+\epsilon_2)$$

där $|\epsilon_k| \leq \epsilon_{mach}$, $k = 1, 2$. Så, om vi multiplicerar ihop faktorerna

$$fl(a+bc) = a + bc + bc\epsilon_1 + (a+bc)\epsilon_2 + bc\epsilon_1\epsilon_2 \Rightarrow$$

$$|fl(a+bc) - (a+bc)| = |bc\epsilon_1 + (a+bc)\epsilon_2 + bc\epsilon_1\epsilon_2|$$

Vi kan nu ge en övre begränsning av det absoluta felet:

$$|fl(a+bc) - (a+bc)| \leq |bc|\epsilon_{mach} + |a+bc|\epsilon_{mach} + |bc|\epsilon_{mach}^2 =$$

$$\left((1+\epsilon_{mach})\,|bc| + |a+bc|\right)\epsilon_{mach}$$

För det relativa felet observerar vi (om $a + bc \neq 0$):

$$\frac{|fl(a+bc) - (a+bc)|}{|a+bc|} \leq \frac{(1+\epsilon_{mach})\,|bc| + |a+bc|}{|a+bc|}\epsilon_{mach} =$$

$$\left[\frac{(1+\epsilon_{mach})\,|bc|}{|a+bc|} + 1\right]\epsilon_{mach}$$

Så det relativa felet är litet om $|bc/(a+bc)|$ inte är för stort. Om däremot $bc \approx 1$, $a + bc \approx 0$, till exempel, kan vi få ett stort relativt fel.

Observera att ett uttryck som $|a\epsilon_1 - b\epsilon_2| \leq (|a| + |b|)\epsilon_{mach}$ även om a och b har samma tecken (ϵ_1 och ϵ_2 kan ju ha olika tecken). Att uppskatta $|(a+b)\epsilon_1| \leq (|a| + |b|)\epsilon_{mach}$ är dock onödigt pessimistiskt, ty a och b kan ju ha olika tecken (notera att det är samma ϵ_1 för båda termerna).

Om man inte kräver en strikt gräns utan endast en uppskattning kan man tillåta sig att slänga t.ex. $\epsilon_1\epsilon_2$-termer (produkter av termer) ty $\epsilon_{mach}^2 \ll \epsilon_{mach}$. Du kan läsa mer om hur man förenklar sådana och andra ϵ-uttryck på sista sidan i övningsmaterialet.

För att se att analysen stämmer rätt bra kommer här ett numeriskt exempel i fyrsiffrig decimal aritmetik:

$$a = 10.70,\ b = -4.567,\ c = 2.344.\ a + bc = -0.005048,\ \text{exakt.}$$

$$fl(a + bc) = fl(a + fl(bc)),\ bc = -10.705048$$

Eftersom $fl(bc) = -10.71$ får vi $fl(a + (-10.71)) = -0.010$. Beloppet av absoluta felet är $|-0.010-(-0.005048)| = 0.004952$. Notera att detta fel är ungefär lika stort som det exakta värdet. Det relativa felet är

$$\frac{0.004952}{0.005048} \approx 0.98.$$

Relativa maskinnoggrannheten, som är 0.0005, har alltså förstorats 0.98/0.0005, en faktor 2000. Vi har inte en siffra rätt i resultatet. Förstoringsfaktorn 2000 stämmer väl med vår analys, som ger:

$$\frac{(1+\epsilon_{mach})\,|bc|}{|a+bc|} + 1 \approx 2123$$

Om vi byter tecken på b, så får vi ingen kancellation, och beräknar en mycket bra approximation av det exakta värdet.

$$a = 10.70,\ b = +4.567,\ c = 2.344.\ a + bc = 21.405048 \text{ exakt.}$$

$$fl(a + bc) = fl(a + fl(bc)),\ bc = 10.705048$$

så

$$fl(bc) = 10.71,\ \text{och } fl(a + 10.71) = 21.41,$$

som är ett korrekt avrundat värde av det exakta resultatet. Det relativa felet är

$$\frac{21.41 - 21.405048}{21.405048} \approx 2.3 \cdot 10^{-4} = 0.46\,\epsilon_{mach}$$

Även detta result stämmer väl med vår uppskattning som ger värdet 0.5 (ungefär).

Talsystemet igen:

Lucka kring nollan. Offra "decimaler" för att få större exponentomfång. Vi använder denormaliserade eller subnormala tal.

```
1.010010011100 ... * 2^e          normaliserat
0.000010100011 ... * 2^(-1022)    denormaliserat
```

Har färre bitar i mantissan. Man talar om "gradual underflow".

Matrisfaktoriseringar

Vanligt i tillämpningar och teoretiskt arbete att skriva matriser som produkter av andra matriser (kallas matrisfaktoriseringar eller uppdelningar). Några exempel illustrerade med små "kryssmatriser":

$$\underbrace{\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} \times & 0 & 0 \\ \times & \times & 0 \\ \times & \times & \times \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \end{bmatrix}}_{U}$$

L för "Lower triangular", undertriangulär och
U för "Upper triangular", övertriangulär.

Kallas LU-faktorisering. Används för att lösa $Ax = b$-problem. Matlab-kommando `lu`.

För att approximativt lösa överbestämda ekvationssystem (minstakvadratproblem) använder vi QR-faktorisering (`qr` i Matlab).

$$\underbrace{\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}}_{Q} \underbrace{\begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \end{bmatrix}}_{R}$$

där Q är ortogonal, dvs. $Q^TQ = I$.

Om A är en sk diagonaliserbar matris kan vi använda Matlabs `eig`-kommando för att beräkna:

$$\underbrace{\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}}_{X} \underbrace{\begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix}}_{\Lambda} \underbrace{\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}}_{X^{-1}}$$

$\lambda_1, \lambda_2, \lambda_3$ är A:s egenvärden och de tre kolonnerna i X är motsvarande egenvektorer. Om A är en reell och symmetrisk matris så kan egenvektorerna väljas ortonormerade varför X är ortogonal och $X^{-1} = X^T$.

Singuläravärdesfaktoriseringen (i Matlab `svd`) är en slags generalisering av egenvärdesuppdelningen till ickekvadratiska matriser.

$$\underbrace{\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}}_{V^T}$$

där $U^TU = I$, $V^TV = I$ och $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq 0$. Faktoriseringen existerar även för liggande matriser.