0 Grundläggande begrepp

0.1 Terminologi: Absolutfel och relativfel

Olika typer av fel uppstår i princip i alla praktiska vetenskapliga resonemang en ingenjör stöter på. Det gäller både traditionella experiementiella fält, men även i beräkningstekniska sammanhang som vi kommer att se i denna kurs.

Vi behöver två grundläggande sätt att kvantifiera (dvs beskriva storleksordningen) fel.

Definition 0.1.1 (Absolutfel)**.** Absolutfel *motsvarar det man i första hand associerar med begreppet fel, dvs skillnaden mellan det exakta värdet x och en approximation $\tilde{x}$*

$$e_x = x - \tilde{x}.$$

Eftersom vi i vanliga fall inte har tillgång till felet brukar man ofta göra analyser med begränsingar av fel. Vi säger att E_x är en absolutfelgräns *om.*

$$|e_x| \leq E_x.$$

Definition 0.1.2 (Relativfel)**.** *Relativfelet ger en indikation på hur bra ett resultat (t.ex. utdata i en simulation) är i relation till resultatets storlek. Vi använder notationen*

$$r_x = \frac{x - \tilde{x}}{x} = \frac{e_x}{x}$$

och kallar R_x för en relativfelgräns *om*

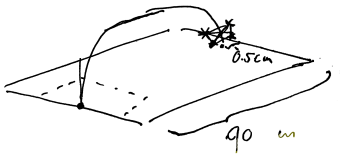
$$|r_x| = \frac{|x - \tilde{x}|}{|x|} \leq R_x.$$

Man kan alltså säga att $R_x = E_x/|x|$ är en relativfelgräns om E_x är en absolutfelgräns. Eftersom x inte är tillgänglig i allmänhet gör man ibland approximationen $R_x \approx E_x/|\tilde{x}|$, och $E_x/|\tilde{x}|$ är en approximativ relativfelgräns (som gäller när e_x inte är för stort).

Exempel: Quadropter

En ingenjör genomför en datorsimulering som bestämmer hur man ska styra motorerna i en quadropter för att den ska flyga över en fotbollsplan (längd 90 m) och landa på mållinjen. När man kör quadroptern landar den cirka 0.5 cm från mållinjen, dvs

$$x = 90 \pm 0.005 \text{ m}. \tag{1}$$



Vi har absolutfel

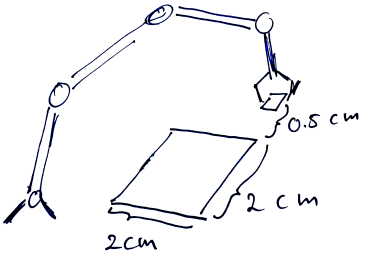
$$e_x \approx 0.005$$

○

Exempel: Robotarm

En robotarm används för att bygga ett kretskort. Styrsystemet i robotarmen bestäms av simuleringar, bland annat genom att lösa differentialekvationer med approximationer. När robotarmen ska placera en känslig komponent som är i position (2, 2) cm från det nedre vänstra hörnet på kretskortet visar det sig att den landar i position (2,2.5) cm. Absolutfelet blir 0.5 cm, och

$$y = 0.02 \pm 0.005 \text{ m}. \tag{2}$$



och

$$E_y = 0.005.$$

Uppdragsgivaren för quadroptern är sannolikt mer tillfredsställd än projektledaren på kretskortföretaget. Detta trots att (1) och (2) har samma absolutfel. Skillnaden mellan de helt olika situationerna fångas bättre med relativfel

$$\begin{aligned} R_x &= 0.005/90 \approx 6 \cdot 10^{-5} \\ R_y &= 0.005/0.02 \approx 3 \cdot 10^{-1} \end{aligned}$$

I de flesta fall inom numeriska beräkningar är relativfel ett mer informativt sätt att kvantifiera kvaltiteten på resultatet än absolutfel.

Båda exemplen ovan är realistiska situationer. Man använder i dagens styrssystem ofta så kallad "Model Predictive Control" (MPC). MPC bygger på att man gör en optimal styrning utifrån en matematisk modell av verkligheten (i form av differentialekvationer som behöver lösas numeriskt).

○

0.2 *Flyttalsaritmetik*

De flesta programmeringsspråk har något sätt att hantera reella tal. Syntaxen i språket brukar vara så att man får oftast får det man förväntar sig av matematiska formler.

Läs: Mer om avrundningsfel och flyttal i Sauer, kapitel 0.3.1
Sauer =
Kursboken, T. Sauer, *Numerical analysis*

```
>> 1.1-1.12
ans =
  -0.020000000000000
```

Programmeringsspråket döljer hur talen representeras och vissa approximationer görs i bakgrunden. Detta händer även när man gör de enklaste operationerna och approximationen blir ibland tydlig

```
>> 1.001-1.0012
ans =
    -2.000000000002000e-04
```

Vi förväntar oss $-2 \cdot 10^{-4}$ men får $-2.000000000002 \cdot 10^{-4}$. Det är uppenbarligen en liten skillnad, men det kan skapa problem.

En dator har bara ändligt minne. Så ärligt talat, kan vi inte förvänta oss att kunna representera alla reella tal på en dator, och behöver därför approximera. Flyttal är det vanligaste sättet att spara (approximationer av) reella tal. Det finns även ett naturligt sätt att göra operationer med dessa flyttal. Detta sker dolt för användaren i högnivåspråk som MATLAB.

Principen bakom flyttal bygger på en kombination av två begrepp ni redan stött på:

- Ni har i andra kurser stött på binära tal, t.ex. heltal skrivna i den binära talbasen
$$101_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5$$
och även hur man kan skriva decimaltal i binär form
$$1.101_2 = 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 1.625$$

- Ni har sedan tidigare stött på den så kallade grundpotensformen (eng. scientific notation) där man skriver tal på formen
$$m \cdot 10^n.$$
Oftast normaliserar talet, dvs man väljer m och n så att $1 \leq m < 10$ och n är ett heltal.

Flyttal är ett sätt att representera tal som bygger *grundpotensformen i det binära talsystemet*.

För att skilja flyttalsaritmetik och vanliga beräkningar som inte ger avrundningsfel, brukar vi kalla vanliga beräkningar för *exakt aritmetik*.

Flyttal

Talet $x = 6.5$ skrivs på normaliserad grundpotensform i den binära talsystemet som
$$1.101_2 \cdot 2^2 = 1.625 \cdot 4 = 6.5.$$

○

Mer exakt betraktar vi tal som kan skrivas som ett binärt tal (med ett givet antal binära siffror $bbb \ldots b$) och en heltalsexponent:
$$\pm 1.bbb \ldots b_2 \times 2^n. \tag{3}$$

Definition 0.2.1 (Flyttal)**.** *Ett flyttal (ibland IEEE-flyttal) är tal som kan skrivas som* (3) *och består av*

- *tecknet:* $\pm$,
- *mantissan:* $m = 1.bbb\ldots b_2$ *som är ett tal i intervallet* $[1, 2)$, *och*
- *exponenten: ett heltal* n.

All information om ett flyttal kan alltså sparas i tecknet, mantissan och exponenten. Antalet siffror som används i mantissan och exponenten bestäms av flyttalets (så kallade) precision. I den standardiserade flyttalsaritmetiken (IEEE-aritmetiken) används någon av följande precisioner:

Den vanligaste flyttalsprecisionen är dubbel precision, som vi använder nästan uteslutande i den här kursen. Se Sauer 0.3.1 för en mer detaljerad beskrivning om hur talet lagras.

- Enkel precision: All information om talet lagras i 32 bittar (4 bytes).
- Dubbel precision: All information on talet lagras i 64 bittar (8 bytes). Mantissan består av 52 binära siffror.
- Lång dubbel precision: All information om talet lagras i 80 bittar.

I samband med ett ökat intresse för inbyggda system (även kallad inbäddade system) har andra flyttalsrepresentationer börjat användas.

På inbyggda enheter, t.ex. quadropters, styrsystem för robotar, grafikprocessorer, app-styrda enheter i hemmet, och annat kopplat till internet-of-things, blir det vanligare och vanligare att använda sig av lägre precision (t.ex halv precision) eftersom detta kräver mindre från hårdvaran. I dessa situationer måste man acceptera att man kan få betydligt större avrundningsfel.

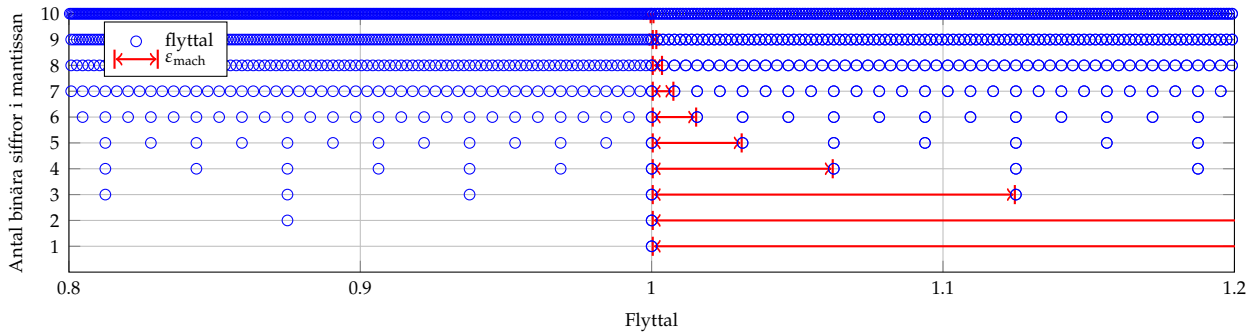
- Halv precision: All information tillhörande talet lagras i 16 bittar (2 bytes).

Med en given precision kan vi inte representera alla tal. För att beskriva finkornigheten av en flyttalsprecision brukar vi jobba med det som kallas för maskinnoggrannhet.

Definition 0.2.2 (Maskinnoggrannhet)**.** *Avståndet mellan 1 och det minsta tal större än 1 som kan representeras i flyttalsprecisionen kallas för maskinnoggrannheten. I dubbel precision är maskinnoggrannheten*

$$\varepsilon_{\text{mach}} = 2^{-52} \approx 2.2 \cdot 10^{-16}.$$

Följande bild illustrerar alla flyttal i närheten av ett, när vi har olika antal binära siffror i mantissan.



Det går att bevisa (men vi gör det ej i denna kurs) att finkornigheten i flyttalen är givet av detta tal. Mer exakt, givet ett reellt tal, avgör maskinnoggrannheten hur långt bort det närmsta flyttalet (i värsta fall) ligger: Om $x \in \mathbb{R}$, så finns det alltid ett flyttal $\tilde{x}$ så att relativfelet är mindre än $\varepsilon_{\text{mach}}/2$, dvs

$$\frac{|x - \tilde{x}|}{|x|} \leq \frac{\varepsilon_{\text{mach}}}{2} \tag{4}$$

Även exponenten har vissa begränsningar för en given flyttalsprecision. Om exponenten blir för stor (för liten) får vi vad som kallas *overflow* (underflow).

Grundläggande operationer med flyttal

Om man adderar två flyttal är det inte alls säkert att resultatet kan sparas som flyttal i samma precision. Man behöver avrunda resultatet. IEEE-flyttalsaritmetiken har en fördelaktig egenskap när det gäller grundläggande aritmetiska operationer. I den standardiserade IEEE-aritmetiken uppfyller det avrundade resultatet av de grundläggande operationerna (plus, minus, gånger, dividerat, mfl) samma felgräns som (4).

Definition 0.2.3 (Relativfel vid flyttalsoperationer)**.** *I IEEE-flyttalsaritmetiken för de grundläggande operationerna beräknas ett resultat som högst skiljer sig åt från det exakta resultatet med en relativ noggrannhet* $\varepsilon_{\text{mach}}/2$.

Exempel

Låt oss säga att vi har en dator med dubbel precision. Om vi beräknar $z = x + y$

$$\begin{aligned} x &= 1.00100_2 \cdot 2^2 = 4.5 \\ y &= 1.00011_2 \cdot 2^3 = 8.75 \end{aligned}$$

så är vi från IEEE-standarden garanterad att få en flyttalsapproximation $\tilde{z} = z$ som uppfyller

$$|z - \tilde{z}| \leq \frac{1}{2}|z|\varepsilon_{\text{mach}} = \frac{13.25}{2} \approx 3 \cdot 10^{-15}.$$

Det kan verka rätt litet. Accumulering av fel kan dock leda till problem.

○

Deterministiskt

När vi adderade våra två tal i föregående exemplet var vi garanterad att resultatet uppfyllde en relativfelgräns. Det finns ofta flera flyttal som uppfyller det villkoret, och IEEE-standarden säger inte exakt vilken approximation vi får. Dock är IEEE-standarden determinstisk i

den bemärkelse att vi alltid får samma flyttalsapproximation. Om vi alltså gör samma sekvens flyttalsoperationer två gånger är vi av IEEE-aritmetiken garanterade att få identiska resultat. I MATLAB får vi till exempel:

```
>> a=sqrt(pi)-exp(1)
>> b=sqrt(pi)-exp(1)
>> z=a-b
z =
   0
```

0.3 *Felfortplantning och felanalys*

Läs: Sauer, kapitel 0.4, 2.3-2.3.2

När vi konstruerar komplicerade numeriska metoder måste vi använda oss av flera approximationer, som genererar olika typer av fel. Vi kommer nu ge några verktyg som hjälper oss att analysera hur dessa fel samspelar, och fel kan propagera genom metoden.

Felpropagering för enkla operationer

Vi betraktar först hur fel propagerar i några enkla operationer. Låt oss säga att $\tilde{x}$ och $\tilde{y}$ är approximationer av x och y och vi har

$$\tilde{x} = x + e_x \quad \tilde{y} = y + e_y.$$

Om vi ser $\tilde{z} = \tilde{x} + \tilde{y}$ som en approximation av $z = x + y$ så har vi

$$|z - \tilde{z}| = |x + y - (x + e_x + y + e_y)| = |e_x + e_y| \leq E_x + E_y.$$

Vi ser att vi kan addera absolutfelgränserna för x och y för att få en absolutfelgräns för summan av x och y. På samma sätt får vi vid subtraktion, dvs $z = x - y$ och $\tilde{z} = \tilde{x} - \tilde{y}$

$$|z - \tilde{z}| = |x - y - (x + e_x - y - e_y)| = |e_x - e_y| \leq |e_x| + |e_y| \leq E_x + E_y.$$

Även vid subtraktion *adderas* felgränserna. För multiplikation har vi

$$\begin{aligned} |xy - x(1 + r_x)y(1 + r_y)| = |xy - xy(1 + (r_x + r_y) + r_x r_y)| \leq \\ |r_x| + |r_y| + |r_x r_y| \leq R_x + R_y + R_x R_y \approx R_x + R_y \end{aligned}$$

där vi antagit att R_x och R_y är små så att $R_x R_y$ är försumbar. Man kan visa motsvarande resultat för division.

Vi har alltså:

- Addition och subtraktion av tal: Addering av absolutfelgränser
- Multiplikation och division av tal: Addering av relativfelgränser (om R_x och R_y små)

Kancellation

Den viktigaste konsekvensen av felpropagering är en situation då väldigt små relativfel propagerar till stora relativfel fel i utdata. Vi kommer nu lära oss om den otrevliga situationen som kallas för *kancellation* (eng. cancellation eller loss of significant digits). Det är framförallt viktigt för flyttalsapproximationer, eftersom vi där är just garanterade att ha bra *relativa* approximationer.

Vi låter återigen $\tilde{x}$ och $\tilde{y}$ vara approximationer av x och y och betraktar relativfel som uppstår vid subtraktion: $z = x - y$ och $\tilde{z} = \tilde{x} - \tilde{y}$. Vi får direkt utifrån definitionerna

$$r_z = \frac{z - \tilde{z}}{z} = \frac{x - y - (x(1 + r_x) - y(1 + r_y))}{x - y} = \frac{x r_x + y r_y}{x - y}.$$

Observera högerledet: Även om r_x och r_y är små, kan r_z bli stort om $x - y$ är litet, dvs om $x \approx y$.

Definition 0.3.1 (Exempel Kancellation 1)**.** *Kancellation uppstår vid subtraktion av två nästan lika tal. Kancellation innebär i regel att små relativfel i termerna leder till ett stort relativfel i differensen.*

Exempel Kancellation 1

Nu ska vi se ett konkret exempel då ett relativfel i storleksordning 0.01 leder till ett relativfel av storleksordning 1.5. Ett relativfel som är större än ett betyder att resultatets osäkerhet är större än talet själv och resultatet är i princip intetsägande.

```
>> x=10; xtilde=10.1;
>> rx=(x-xtilde)/x

  -0.010000000000000
>> y=9.9; ytilde=9.85;
>> ry=(y-ytilde)/y
ry =
   0.005050505050505
>> z=x-y; ztilde=xtilde-ytilde;
>> rz=(ztilde-z)/z
rz =
   1.500000000000009
```

Exempel Kancellation 2

I följande exempel gör vi matlab beräkningar med enkel aritmetik, med värden $x = 1.1$ och $d = 10^{-5}$. Vi ska jämföra beräkningen via två formler, högerledet och vänsterledet i

$$\frac{\sqrt{x+d}-\sqrt{x}}{d} = \frac{1}{\sqrt{x+d}+\sqrt{x}}.$$

Man kan visa att högerledet och vänsterledet är lika med hjälp av identiteten $(\sqrt{x+d}-\sqrt{x})(\sqrt{x+d}+\sqrt{x}) = x+d-x = d$. Hur blir det i flyttalsaritmetik?

```
>> (sqrt(x+d)-sqrt(x))/d
ans =
   0.4768372
>> 1/(sqrt(x+d)+sqrt(x))
ans =
   0.4767302
```

Observera att talen är lika i (vanlig) exakt aritmetik, men ger olika resultat i flyttalsaritmetik. Den bästa är den andra, eftersom den inte innehåller någon subtraktion och alltså inte har kancellation.

○

Allmänna felfortplantningsformeln

I teorin kan man använda analysen ovan för att studera metoder bestående av de grundläggande operationerna. I praktiken blir det snabbt komplicerade beräkningar och inte så insiktsfullt. För mer komplicerade metoder använder man istället en formel för felpropagering, som gäller som uppskattning om felen är tillräckligt små.

Om F är en funktion som kan deriveras tillräckligt ofta (t.ex. om den består av en kombination av grundläggande operationer) har vi från Taylorutvkeckling

$$y = F(x) = F(\tilde{x}-e_x) = F(\tilde{x}) - e_x F'(\tilde{x}) + e_x^2 F''(\tilde{x}) + \cdots$$

Låt oss säga att vi vill se hur ett fel i x propagerar till ett fel i y, där vi approximerar $\tilde{y} = F(\tilde{x})$. Om $e_x \ll 1$ så har vi uppskattningen

$$\tilde{y} - y = e_y \approx e_x F'(\tilde{x}),$$

eller E_y är en approximativ felgräns till $\tilde{y}$ om vi sätter

$$E_y = E_x |F'(\tilde{x})|.$$

Mer allmänt, för en funktion i flera variabler $y = f(x_1, \ldots, x_n)$ med approximationen $\tilde{y} = f(\tilde{x}_1, \ldots, \tilde{x}_n)$ med har vi den Allmänna Felfortplantningsformeln

$$E_y \approx E_{x_1}\left|\frac{\partial F}{\partial x_1}\right| + \cdots E_{x_n}\left|\frac{\partial F}{\partial x_n}\right|$$

Ett exempel på allmänna felfortplantningsformeln finns i Exempel 3 här http://www.kth.se/social/files/563795e5f276547b53df41e6/Ant-Felanalys.pdf.

där $\partial F/\partial x_i$ är den (så kallade) partiella derivatan, som motsvarar derivering av F med avseende på x_i, där alla variabler $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$ betraktas som konstanta.

Partiella derivator och Taylor-utveckling i flera variabler kan ni lära er mer om i kursen i flervariabelanalys.

I detta sammanhang kan man även betrakta F som problemet i sig, och man brukar kalla

$$|xF'(x)/F(x)|$$

för problemets konditionstal. Konditionstalet är en kvantifiering på hur små störningar i indata påverkar resultatet. Stort konditionstal betyder att problemet i sig är väldigt känsligt för förändringar i indata. Detta är en problemegenskap (inte metodegenskap) och en numerisk metod har ingen chans att ge noggranna resultat från ett dåligt konditionerat problem.

Läs mer om konditionstal för matriser och ekvationssystem i Sauer 2.3.

0.4 *Beräkningskostnad (för Gauss-eliminering)*

Den här kursen handlar om *tillförlitliga* och *effektiva* beräkningar. Felanalysen ovan är en teknik för att kvantifiera tillförlitligheten. Hur kan vi kvantifiera hur effektiv en metod är?

Det vanligaste sättet att kvantifiera effektiv en metod är (dvs om den kan genomföras med lite datortid) är att räkna antalet flyttalsoperationer som krävs för att genomföra metoden. Ni har lärt er om Gauss-eliminering i andra kurser. Gauss-eliminering är en metod som man kan använda för att lösa linjära ekvationssystem

En mer exakt härledning av antalet flyttalsoperationer som krävs för Gauss-eliminering finns beskrivet i Sauer

Läs: Sauer, kapitel 2.1-2.1.2 (2.1.1)

$$Ax = b$$

där $A \in \mathbb{R}^{n\times n}$ och $b \in \mathbb{R}^n$ är givna och $x \in \mathbb{R}^n$ ska beräknas. Den enklaste varianten av Gauss-eliminering för en matris $A \in \mathbb{R}^{n\times n}$ kräver såhär många flyttalsoperationer för att göra den triangulära reduktionen

$$p(n) = \frac{2}{3}n^3 + \frac{1}{2}n^2 - \frac{7}{6}n$$

För några årtionden sedan kunde man nästan exakt förutsäga hur lång beräkningstid man behöver för en given matris med hjälp av formeln $p(n)$. I och med utvecklingen av datorer, datorarkitekturer, processorer och kompilatoroptimering är så inte längre fallet. Formeln ger dock viss insikt för effektiviteten framförallt för stora matriser n. När vi uppskattar beräkningstid brukar istället skriva endast den termen som blir dominant när n är stort och ignorera koefficienten,

I matlab finns en operator (backslash-operatorn: `A\b`) som kan användas för att lösa linjära ekvationssystem och bygger på en väldigt optimerad variant av Gauss-eliminering. Obs: A måste ha lika många rader som kolumner. Om A har fler rader än kolumner existerar inte A^{-1}.

$$p(n) = \mathcal{O}(n^3).$$

Notationen med $\mathcal{O}$ kallas för ordo-notation. Ibland säger vi att komplexiteten är n^3.

Man kan även visa (se Sauer) att bakåtsubstitutionsdelen av Gausseliminering kräver

$$p(n) = \mathcal{O}(n^2)$$

flyttalsoperationer.