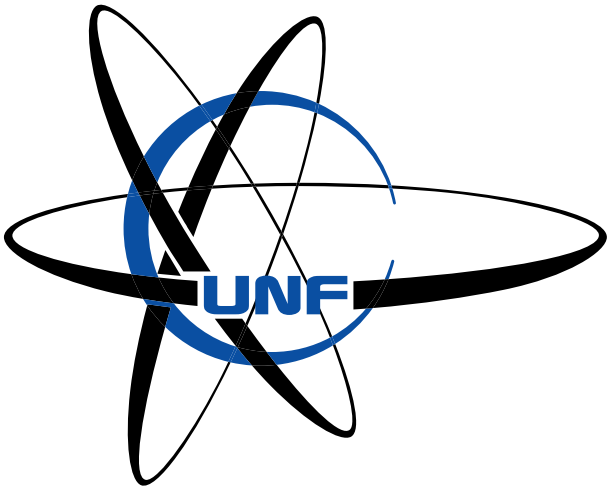


UNF Naturfagsweekend 2022

UNF København

Faglige:

Rasmus Frigaard Lemvig	rle@unf.dk
Marie Stuhr Kaltoft	mark@unf.dk
Erik Søndergård Gimsing	esg@unf.dk
Frederick Aleksander Nilsen	fran@unf.dk
Louie Ray Eistrup Juhl	loue@unf.dk
Jinyang Liu	jili@unf.dk
Knut Ibæk Topp Lindenhoff	knut@unf.dk
Lin Bigom-Eriksen	libe@unf.dk
Mette Kjærgaard Thorup	mkth@unf.dk
Robert Garbrecht Larsen	roe@unf.dk
Niels Anders Lyngsø Bærentzen	nalb@unf.dk
Ask Kildelund Rosenkilde	akr@unf.dk
Simone Bidsted	sibi@unf.dk
Sofie Krogsgaard	skr@unf.dk
Catarina Nampumee Maretti	cnm@unf.dk
Christine Ibæk Topp Lindenhoff	cit@unf.dk
Mikkel Møller Mødekjær	mim@unf.dk

Ungdommens Naturvidenskabelige Forening

Kompendium til UNF Naturfagsweekend 2022

Kompendiet er skrevet af Rasmus Frigaard Lemvig, Marie Stuhr Kaltoft, Erik Søndergård Gimsing, Frederick Aleksander Nilsen, Louie Ray Eistrup Juhl, Jinyang Liu, Knut Ibæk Topp Lindenhoff, Lin Bigom-Eriksen, Mette Kjærup Thorup, Robert Garbrecht Larsen, Niels Anders Lyngsø Bærentzen, Kiro Kildelund Rosenkilde, Simone Bidsted, Sofie Krogsgaard, Catarina Nampumee Maretti, Christine Ibæk Topp Lindenhoff, Mikkel Møller Mødekjær og Thomas Emil Le Cozannet. Kompendium er trykt i februar 2022,

Layout: Esben Skovhus Ditlefsen på forarbejde af Niels Jakob Søe Loft og Mick Althoff Kristensen.

Opsætning/TEXnisk ansvarlig: Morten Raahauge Bastholm.

Indhold

Introduktion

Velkommen til det faglige kompendium! Kompendiet er bygget op af seks kapitler, som dækker de seks forløb, der er på campen. Hvert kapitel er udarbejdet af de faglige teams for hvert område. De samme teams står også for undervisningen på campen, hvor kompendiet benyttes som det primære undervisningsmateriale. Kompendiet er dog også velegnet til selvstudie. Alle os faglige ønsker jer god fornøjelse med campen!

Derudover ønsker vi at rette en stor tak til NEXT Sukkertoppen Gymnasium, DUFs lokalforeningspulje, Novataris og FLSmidth gavefond for at muliggøre campen. Der står mere information om sponsorerne for campen bagerst i kompendiet.

Kapitel 1

Matematik

1.1 Introduktion

Velkommen til matematikforløbet på Naturfagsweekend! Inden vi springer ud i hovedemnet for forløbet, vil vi gerne give en lille introduktion til, hvad matematik er for en videnskab. Hvad tænker du på, når du hører ordet "matematik"? Du tænker sikkert på regning med tal, optegning af figurer med lineal eller passer osv. Men det er faktisk slet ikke essensen i matematikken. Matematikere er ofte slet ikke gode til regning! Matematikere er til gengæld gode til at tænke abstrakt og løse problemer med meget specifikke værktøjer. Det er i arbejdet med udviklingen af disse værktøjer, at det sjove og interessante ved matematikken fremkommer. Derudover har matematik også en filosofisk side. Herunder har vi forsøgt at give en beskrivelse af, hvad matematik er for os.

Hvad er matematik for os?

Marie

I modsætning til de andre videnskaber, så kan man altid stole på de resultater, som vi finder i matematik. Jeg har altid været utrolig interesseret i naturvidenskab og specielt matematik. Da jeg selv gik i folkeskole var jeg overbevist om, at jeg ville læse fysik, men senere gik det op for mig, at det faktisk var det matematiske, der primært interesserede mig ved fysikken. Det var netop denne sikkerhed ved matematikken som jeg fandt fascinerende. En sikkerhed som jo ikke findes på samme måde andre steder. Når jeg generelt tænker på matematik, så tænker jeg ikke længere på det, som man laver i folkeskolen eller gymnasiet for den sags skyld. Jeg tænker på det, som en abstrakt måde at angribe problemer på. Her er vores emne om grafteori et rigtig godt eksempel på, hvordan man kan bruge abstrakt matematik til at angribe helt konkrete problemer, som vi også vil komme ind på i forløbet.

Erik

Den korte version er, at matematik er hvad matematikere laver. Det vil sige, der er som så ikke nogle regler for, hvad der er matematik, men derimod en række krav til den metode, der anvendes i arbejdet. Det skal være logisk, helt logisk. Det skal være bevisbart sandt, det vil sige du skal kunne lave et argument, der er overbevisende og sandt. Så matematik er en metode til at tænke over ting, hvor man tager sin ide og gør den abstrakt og logisk, og derefter undersøger den ved at argumentere for diverse egnskaber. På den måde er matematik ikke en empirisk videnskab, men heller ikke humaniora, da alt hvad matematik kommer frem til er utvetydig sandt givet nogle antagelser. Det er hvad jeg nyder ved det, jeg kan tænke over nogle ting jeg finder interessante og være helt sikker på, at det jeg finder ud af er sandt. Det jeg bedst kan lide er at omsætte et spørgsmål jeg har til matematik, og så finde værktøjer til at svare på det.

Rasmus

Matematik for mig er en abstrakt videnskab, hvor fremgangsmåden er meget grundig argumentation. Man kan sige, at vi laver beviser, hvor man i naturvidenskab laver forsøg. Når noget i matematik er bevist, kan man være helt sikker på, at det er sandt. Her adskiller matematik sig fra f.eks. fysik og biologi, hvor nye forsøg kan ændre hele videnskaben, og en masse ting må gøres om. Jeg betragter det at arbejde med matematik som, at vi laver nye værktøjer, der kan indgå i mange forskellige sammenhænge.

Selve forløbet

I dette forløb skal vi arbejde med grafteori, som er en gren af såkaldt diskret matematik (matematik uden "kommatal"). Hvad skal I få ud af forløbet? Vi håber naturligvis på, at I bliver klogere, men det er faktisk vigtigere, at I udvider jeres horisont og opdager, at matematik er langt mere end bare regning. Synes du ikke, at matematik er det sjoveste fag i dagligdagen, kan faget stadig være noget for dig! Den eneste ting, du skal kunne fra starten i dette forløb er at lægge (relativt små) tal sammen, og så skal du være nysgerrig på at lære noget nyt.

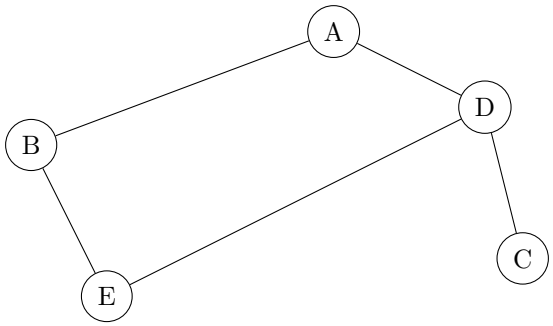
1.2 Hvad er grafteori?

Grundlæggende begreber

Grafteori er studiet af grafer. Du tænker måske på grafer af funktioner i et koordinatsystem, men dette er en helt anden type graf. En graf i dette forløb er en samling af knuder og kanter, der forbinder knuderne med hinanden. Vi lader dette være en definition.

Definition 1.2.1. En graf G er en samling af *knuder* og *kanter*, der går fra en knude til en anden. Vi lader $n(G)$ betegne antallet af knuder i grafen.

Lad os se på et eksempel:



Figur 1.1: Eksempel på en graf

Knuder tegner vi som cirkler eller firkanter, som kan have et navn (her blot bogstaver), mens kanterne er linjerne, der forbinder knuderne. Man kan tænke på grafer på flere måder. Forestil dig f.eks. et vejkort. Her er knuderne placeringer og kanterne veje. Man kan i det hele taget tænke på grafer som illustrationer af ting, der er relateret til hinanden. Grafteori har mange konkrete anvendelser, nok fordi man kan tænke på grafer på flere måder. Dette har nok også været motivationen for opfindelsen (eller opdagelsen?) af dette felt af matematikken. Grafteoriens grundlægger er en af de største matematikere nogensinde, Leonhard Euler. Han undersøgte et konkret problem angående broerne i Königsberg (i dag hedder byen Kaliningrad og ligger i Rusland), nemlig om det er muligt at gå over hver af de 7 broer over floden Pregel (nu kaldet Pregolya) præcist én gang [1]. Lidt senere i forløbet får I lov til at gå i Eulers fodspor og løse dette problem.

I dette afsnit skal vi lære de mest grundlæggende begreber i grafteori, så vi er klar til at løse rigtige problemer i de senere afsnit. Der kan være en del begreber at huske, men heldigvis er de fleste af navnene ikke mystiske.

Definition 1.2.2. Vi har følgende definitioner:

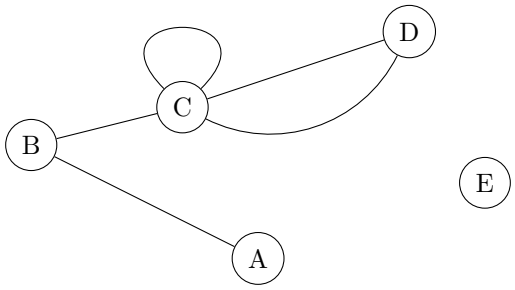
1. En *løkke* er en kant, som starter og slutter i samme knude.

2. To forskellige knuder kaldes *naboknuder*, hvis der er en kant, der forbinder dem.

3. *Valensen* af en knude er lig antal kanter, der har endepunkt i knuden (en løkke lægger 2 til valensen). Den *totale valens* for en graf er summen af valensen for alle knuder i grafen. En knude med lige valens kaldes for en *lige knude*, og tilsvarende kaldes en knude med ulige valens for en *ulige knude*. *Maksimalvalensen* for en graf er den højeste valens, der forekommer i grafen, og *minimalvalensen* er den mindste valens, der forekommer.

4. En *isoleret* knude er en knude, hvor ingen kanter har endepunkt. Med andre ord, en knude med valens 0.

Lad os illustrere disse begreber med et eksempel:



Figur 1.2: Illustration af begreberne i definition 1.2.2. A og B er et eksempel på to naboknuder. C har en løkke. E er en isoleret knude, idet denne knude ingen kanter har. Valensen af A er 1, for B er den 2. Valensen af C og D er henholdsvis 5 og 2. Bemærk, at E har valens 0, da det er en isoleret knude. Altså er maksimalvalensen 5, og minimalvalensen er 0.

Når man i matematik har indført en definition, vil man ofte gerne bygge ovenpå definitionerne med nogle resultater. Valens er et vigtigt begreb, som vi kommer til at bruge en del fremover. Vi har denne sætning om valensen i en graf:

Sætning 1.2.3. *Den totale valens for en graf er lig antallet af kanter gange 2.*

Bevis. Alle kanter i en graf starter og slutter i en knude (muligvis den samme knude). Altså må alle kanter bidrage med 2 til grafens totale valens. ■

Når man har vist en sætning i matematik, kan det ske, at man får et nyttigt resultat eller to mere herudfra. Sådan en følgesætning kaldes et *korollar*. Vi har sådan en hjælpesætning:

Korollar 1.2.4. *Den totale valens for en graf er et lige tal.*

Bevis. Da den totale valens for en graf er lig 2 gange antallet af kanter, må 2 gå op i den totale valens. Altså er den totale valens lige. ■

Når vi arbejder med grafer, vil vi gerne kunne beskrive, hvordan forskellige knuder er relateret til hinanden i grafen. Derfor indfører vi nogle flere begreber:

Definition 1.2.5. Lad G være en graf med knuder A og B.

1. Der er en *rute* mellem A og B, hvis vi kan komme fra A til B ved at bevæge os langs nogle knuder og kanter i grafen. Følgen af knuder og kanter, vi bevæger os igennem, er selve ruten.

2. En *tur* fra A til B er en rute fra A til B, hvor man kun må gå langs hver kant én gang.

3. En *vej* fra A til B er en tur fra A til B, hvor man kun må passere hver knude én gang.

4. En rute kaldes *lukket*, hvis første og sidste knude på ruten er ens. Samme gælder for en lukket tur og en lukket vej. Hvis en rute, tur eller vej ikke er lukket, kaldes den *åben*.

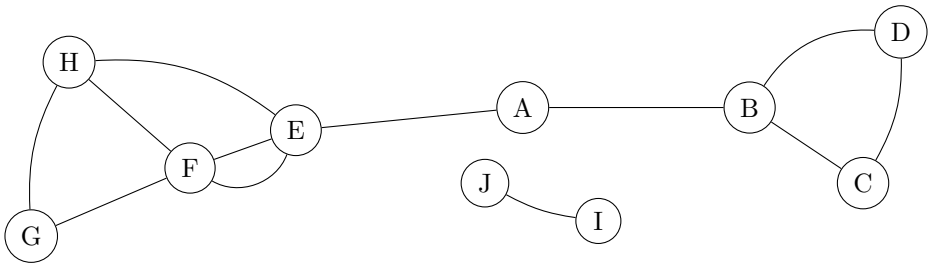
5. En *kreds* er en lukket tur, hvor de eneste gentagne knuder er start- og slutknuden.

For at gøre definitionerne lidt lettere at huske, har vi følgende tabel:

Tabel 1.1: Tabel til begreberne i definition 1.2.5

Definition	Gentagne kanter	Gentagne knuder	Samme start- og slutknude
Rute	Tilladt	Tilladt	Tilladt
Tur	Nej	Tilladt	Tilladt
Vej	Nej	Nej	Nej
Lukket rute	Tilladt	Tilladt	Ja
Lukket tur	Nej	Tilladt	Ja
Kreds	Nej	Kun første og sidste	Ja

Ser vi på figur 1.2, kan vi se, at der er ruter mellem alle knuder undtagen E. Der findes også nogle lukkede ture. F.eks. kan vi gå fra D til C langs én af kanterne og tilbage til D gennem den anden kant. Denne tur er også en kreds. Kan du finde andre lukkede ture? Andre kredse? En anden illustration af begreberne ses i grafen herunder:



Figur 1.3: En graf med mange slags ruter.

I denne graf er der mange ruter! Dog er der ikke ruter mellem alle knuder, f.eks. findes ingen rute mellem I og A. Vi kan ligeledes finde mange ture, også lukkede af slagsen. F.eks. kan vi gå fra B til C til D og tilbage til B. Dette er også en kreds. Kan vi finde en lukket tur, som ikke er en kreds? Ja, lad os starte i F og gå til G, til H, tilbage til F, til E og tilbage til F langs den anden kant. Dette er en lukket tur, da ingen kanter bliver brugt mere end én gang, og vi både starter og slutter i F. Men vi passerer F undervejs i turen, så F optræder ikke kun som start- og slutknude. Derfor er denne lukkede tur ikke en kreds.

Se igen på figur 1.3. Der er noget særligt ved knuderne I og J i forhold til de andre knuder i grafen. I og J kan forbindes med en rute, men de kan ikke forbindes med ruter til nogle andre knuder i grafen. På samme måde kan knuderne fra A til H forbindes med ruter indbyrdes. Disse overvejelser giver os ideen til en ny definition:

Definition 1.2.6. En graf kaldes *sammenhængende*, hvis det for alle par af knuder i grafen gælder, at disse kan forbindes med en rute.

Graferne i figur 1.2 og 1.3 er ikke sammenhængende. Det er grafen på figur 1.1 til gengæld. Vi kommer primært til at arbejde med sammenhængende grafer.

Definition 1.2.7. Hvis A er en knude i en graf G, så består *sammenhængskomponenten*, som indeholder A, af alle de knuder, som er forbundet til A via en rute, samt alle kanterne mellem disse knuder.

Graferne i figur 1.2 og 1.3 består hver især af to sammenhængskomponenter. Grafen i figur 1.1 har kun én sammenhængskomponent.

Nogle graftyper

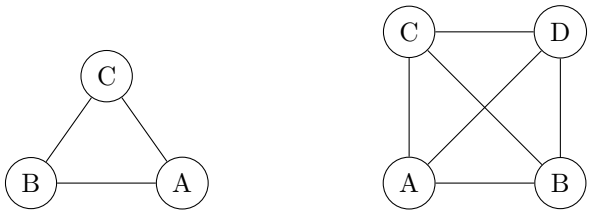
Vi vil nu komme ind på nogle bestemte typer grafer, som er gode at kende.

Definition 1.2.8. En *simpel graf* er en graf uden løkker, og hvor ingen naboknuder er forbundet af mere end én kant.

Grafen i figur 1.1 er et eksempel på en simpel graf, mens grafen i figur 1.2 ikke er en simpel graf. F.eks. har knude C en løkke. Man kan tænke på simple grafer som grafer uden "overflødige kanter", altså der er netop det antal kanter i grafen, som skal bruges, for at de samme knuder er forbundet. Vi ser nu på en speciel type af simple grafer.

Definition 1.2.9. En *komplet graf* er en simpel graf, hvori alle par af forskellige knuder er forbundet med hinanden. Den komplette graf med n knuder betegner vi som K_n.

Lad os se på de to komplette grafer K_3 og K_4 som eksempel:

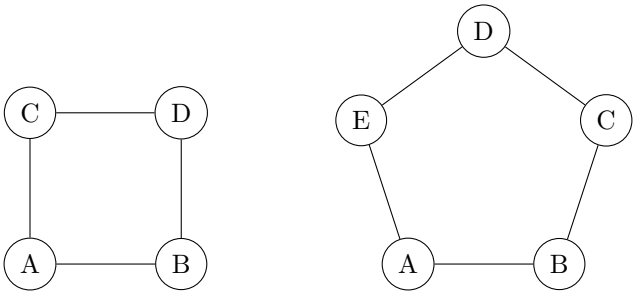


Figur 1.4: Graferne K_3 og K_4.

Den sidste type graf, vi vil kigge på her, er kredsgrafer. Definitionen er meget, som navnet antyder.

Definition 1.2.10. En *kredsgraf* er en simpel graf, hvor alle knuderne indgår i netop én kreds. Kredsgrafen med $n \geq 3$ knuder betegner vi som C_n (C'et står for "cycle", som betyder "kreds" på engelsk).

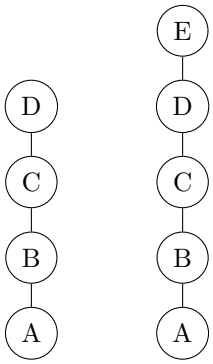
Hvorfor skal vi have $n \geq 3$? Hvis n f.eks. er 2, kan vi slet ikke lave en kreds uden at skulle tegne flere kanter mellem de to knuder. Men så kan grafen ikke være simpel! Ligeledes går det også galt i tilfældet $n = 1$. Definitionen giver altså kun mening, hvis $n \geq 3$. Lad os se på et eksempel, nemlig graferne C_4 og C_5:



Figur 1.5: Graferne C_4 og C_5.

Definition 1.2.11. En *vejgraf* er en simpel graf, hvor én vej kan gennemløbe alle knuder og kanter i grafen. Vejgrafen med n knuder betegner vi P_n (P'et står for "path", som er det engelske ord for "vej").

Nedenfor ser vi to eksempler på vejgrafer.



Figur 1.6: Graferne P_4 og P_5.

Vi kan afslutte med at bemærke, at komplette grafer, kredsgrafer og vejgrafer (de tre standardgrafer) altid er sammenhængende. Vi er nu parate til at kaste os ud i at løse nogle interessante problemer med grafteori.

Opgaver til Hvad er grafteori?

- **Opgave 1.2.1:**
Tegn følgende:

  1) En graf med tre knuder med valens 1, 2 og 3.

  2) En graf med fire knuder alle med valens 1.

  3) En graf med fem knuder med valens 1, 1, 2, 2 og 4.

  4) En ikke-sammenhængende graf med fire knuder, alle med valens 2.

  5) Hvor mange sammenhængskomponenter er der i hver af de grafer, som du har tegnet?

- **Opgave 1.2.2:**
Tegn dit hus/din lejlighed som en graf, hvor et værelse er en knude, og kanterne vejene mellem disse. Hvordan kan man f.eks. håndtere flere etager?

- **Opgave 1.2.3:**
Tegn en sammenhængende graf uden løkker, der indeholder netop 2 kredse. Find maksimal- og minimalvalensen.

- **Opgave 1.2.4:**
Hvor mange grafer kan du tegne, som er sammenhængende, simple, og hvor alle knuder har lige valens? Kender du nogle i forvejen? Hvor mange findes der?

- **Opgave 1.2.5:**
Tegn graferne:

  1) Den komplette graf med 5 knuder, K_5. Find maksimal- og minimalvalensen.

  2) Kredsgrafen med 6 knuder, C_6. Find maksimal- og minimalvalensen.

  3) En sammenhængende graf med 7 knuder. Knuderne skal have valens 1, 2, 3, 4, 5, 6 og 7.

- **Opgave 1.2.6:**
Tegn en graf med 4 knuder. Knuderne skal have valens 1, 2, 2 og 3. Findes en graf med fire knuder af valens 1, 2, 2 og 4? Hvis ja, tegn den. Hvis nej, hvorfor ikke?

- **Opgave 1.2.7:**
Se på K_n og C_n. For hvilke værdier af n er disse grafer ens? Husk, at $n \geq 3$. Hvad er maksimal- og minimalvalensen for disse grafer?

- **Opgave 1.2.8:**
I disse opgaver skal vi se på, om det er muligt i en gruppe af personer, at hver person er ven med et bestemt antal personer i gruppen. Vi antager, at hvis person x er ven med person y, er person y også ven med person x. Hvis det er muligt, tegn da en graf, som illustrerer vennerelationerne.

  1) Antag, at gruppen er på 4 personer. Er det muligt for alle gruppemedlemmer at være venner med præcist 2 andre i gruppen?

  2) Antag, at gruppen er på 7 personer. Er det muligt, at alle gruppemedlemmer er venner med præcist 5 andre i gruppen?

  3) Antag, at gruppen er på 57 personer. Er det muligt, at alle gruppemedlemmer er venner med præcist 33 andre i gruppen?

●● **Opgave 1.2.9:**
Vis, at der i alle grafer er et lige antal knuder med ulige valens.

●● **Opgave 1.2.10:**
Vis, at alle sammenhængende grafer med n knuder har mindst $n-1$ kanter.

●●● **Opgave 1.2.11:**

1) Vis, at K_n har $\frac{n(n-1)}{2}$ kanter [Vink: Tegn K_n for f.eks. $n = 5$ og tæl op for hver kant, brug dette til at give et argument for det generelle tilfælde].

2) Vis, at en simpel graf med n knuder højst kan have $\frac{n(n-1)}{2}$ kanter.

3) Find en simpel graf, der har dobbelt så mange kanter, som den har knuder [Vink: brug første delopgave].

●●● **Opgave 1.2.12:**
I denne opgave indfører vi et nyt begreb, nemlig et *træ*. Et træ er en sammenhængende graf uden nogle kredse.

1) Tegn et træ med henholdsvis 3, 5 og 10 knuder.

2) Giv et argument for, at et træ bliver nødt til at være simpelt.

3) Hvor mange træer findes der med 4 knuder? Hvad med 5 knuder?

4) Findes et træ med 7 knuder og 7 kanter? Hvis ja, tegn træet. Hvis nej, hvorfor?

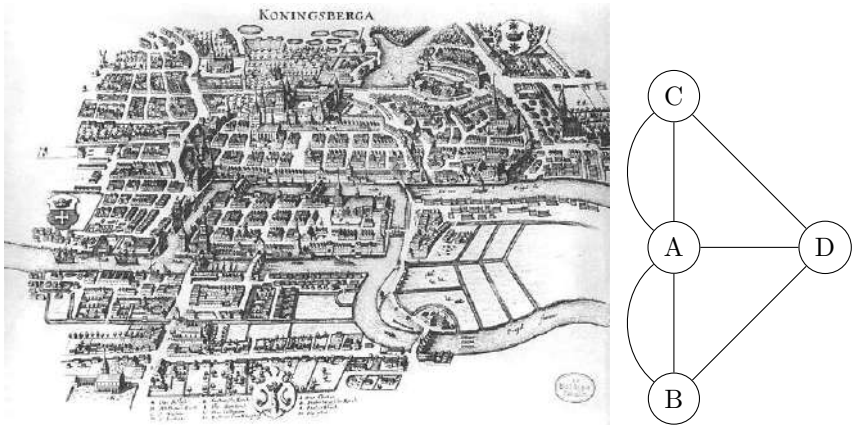
5) Hvor mange kanter er i et træ med n knuder? [Vink: brug opgave 1.2.10. Hvad sker der, hvis man tilføjer flere kanter?]

6) Hvad er den totale valens for et træ med n knuder?

1.3 Flere begreber og egenskaber for grafer

Historien om Königsberg

I 1736 blev Leonhard Euler bedt om at finde en rute gennem byen Königsberg, så en procession (et optog) kunne gå over samtlige syv broer i byen præcis én gang (som I måske husker fra tidligere, er det definitionen af en "tur"). Året efter udgav han en artikel, som analyserede, hvorvidt dette var muligt. Denne artikel betragtes som det første bidrag til grateorien [2]. Nedenfor på figur 1.7 ses et billede af Königsbergs broer, og hvordan disse kan repræsenteres ved en graf.



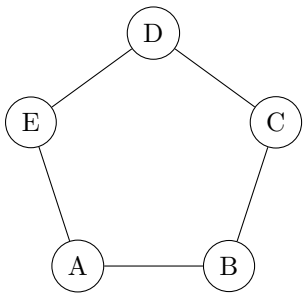
Figur 1.7: *Venstre*: Tegning af Königsberg med broerne [3]. *Højre*: Königsberg repræsenteret ved en graf. Knuderne er de fire landområder, og kanterne er broerne imellem dem.

Efter dette store bidrag til grafteorien har Euler naturligvis fået opkaldt et begreb efter sig:

Definition 1.3.1. En *Euler-tur* er en tur, som indeholder alle grafens kanter.
En Euler-tur kan enten være åben eller lukket. Her genbruger vi blot definition 1.2.5-(4):

- En lukket Euler-tur starter og slutter i samme knude.
- En åben Euler-tur starter og slutter i to forskellige knuder.

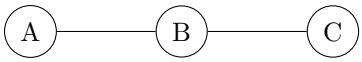
Eksempel 1.3.2. For et helt simpelt eksempel på en lukket Euler-tur kan vi se på kredsgrafen med 5 knuder igen:



For at lave en lukket Euler-tur kan vi vælge en vilkårlig knude at starte i. For eksempel kan vi starte i A. Vi kan nu gå turen $A \to B \to C \to D \to E \to A$. Da alle kanterne er med i turen netop én gang og vi har samme start- og slutknude, har vi altså lavet en lukket Euler-tur i grafen!

Overvej: Gælder det for alle kredsgrafer? ○

Eksempel 1.3.3. For et tilsvarende eksempel på en åben Euler-tur kan vi se på vejgrafen med 3 knuder P_3:



For at lave en åben Euler-tur skal vi vælge en knude at starte i med en smule omtanke. Vi skal nemlig starte og ende i en af endeknuderne for, at vi kan få alle kanter og knuder med. For eksempel kan vi starte i A. Vi kan nu gå turen $A \to B \to C$. Da alle kanterne er med i turen netop én gang, og vi har forskellige start- og slutknuder, har vi altså lavet en åben Euler-tur i grafen! ○

Som I måske har bemærket, kan det blive svært at forudsige, om en graf har en Euler-tur, når den har mange knuder og kanter. Faktisk er der et relativt simpelt trick, som kan afgøre om en graf har en Euler-tur!

Sætning 1.3.4. *Betragt en graf G uden isolerede knuder.*

1. *G har en lukket Euler-tur, hvis og kun hvis den er sammenhængende og alle knuder har lige valens.*
2. *G har en åben Euler-tur, hvis og kun hvis den er sammenhængende og har præcist to knuder med ulige valens.*

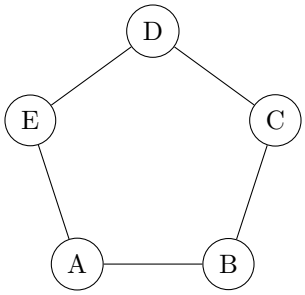
Bevis. For at bevise påstand 1, skal vi vise to ting. Første skal vi vise, at hvis G har en lukket Eulertur, er G sammenhængende, og alle knuder har lige valens. Dette overlades som en øvelse, se opgaverne. Dernæst skal vi vise, at hvis en graf G er sammenhængende, og alle knuder har lige valens, vil G have en lukket Eulertur. Denne implikation udelades, da den bygger på et induktionsargument, som rækker en del udover forløbets indhold. At bevise påstand 2 forløber på tilsvarende vis. ■

Komplementer, kliker og uafhængige delmængder

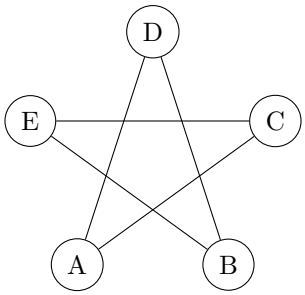
Definition 1.3.5. Lad G være en simpel graf. *Komplementet* til G er grafen med samme knuder som G, og hvor to knuder er naboer hvis og kun hvis de ikke er naboer i G. Vi betegner komplementet af G som $\overline{G}$ (vi plejer at udtale $\overline{G}$ som "G bar").

Komplementet af en simpel graf er altså grafen, som indeholder alle de "manglende" kanter, der skal til for at få en komplet graf.

Eksempel 1.3.6. Betragt grafen G herunder:



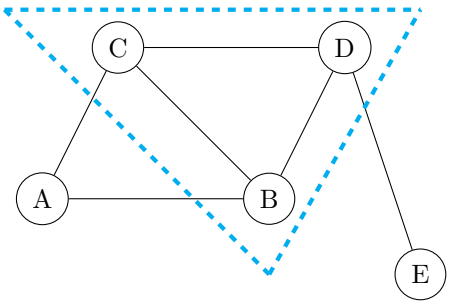
Komplementet til G bliver



Læg mærke til, at hvis man tager disse to grafer og lægger dem "oven i hinanden", så får man den komplette graf med fem knuder, K_5. ∘

Til slut i forløbet skal vi se på farvelægning af grafer. I den forbindelse er det smart at kende til kliker og uafhængige delmængder, specielt når vi skal se på det såkaldte *kromatiske tal* for grafer.

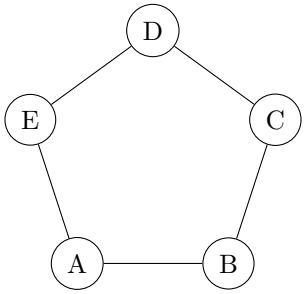
Definition 1.3.7. Lad G være en simpel graf. En *klike* i G er en samling af knuder, som er parvist naboer. En *uafhængig delmængde* er en samling af knuder, som parvist ikke er naboer.



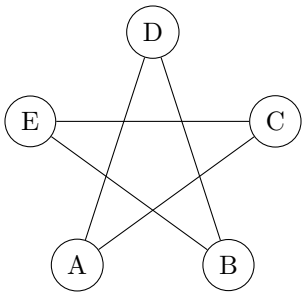
Figur 1.8: B, C, D udgør en klike.

Bemærk, at en klike og en uafhængig delmængde i en simpel graf på en måde er omvendte begreber. Vi kan faktisk sige dette mere præcist, hvilket vi gør om et øjeblik. Først skal vi have nogle eksempler.

Eksempel 1.3.8. Lad os igen se på grafen fra eksempel 1.3.6:



Vi ser, at A og B udgør en klike. Det samme gælder om B og C, C og D og så videre. Vi kan faktisk se, at en klike ikke kan have flere end to knuder i dette eksempel. Vi ser, at A og C udgør en uafhængig delmængde, ligesom A og D gør det. Vi bemærker også, at en uafhængig delmængde maksimalt kan indeholde to knuder. Ser vi på komplementet



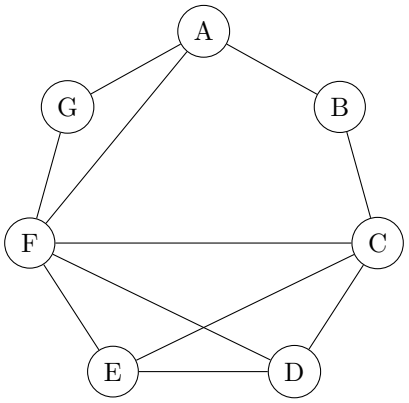
kan vi se, at B, C og C, D bliver uafhængige delmængder, og at A, C og A, D bliver kliker. At kliker bliver til uafhængige delmængder i komplementet og omvendt, gælder generelt, hvilket vi viser herunder. ∘

Sætning 1.3.9. *En klike i G bliver til en uafhængig delmængde i komplementet $\overline{G}$, og en uafhængig delmængde i G bliver til en klike i $\overline{G}$.*

Bevis. Lad knuderne $v_1, ..., v_n$ udgøre en klike i G. Da gælder, at alle knuderne $v_1, ..., v_n$ er parvist naboer, altså at de hver især har en kant til alle andre knuder. Altså vil disse knuder parvist ikke have kanter imellem sig i komplementet, med andre ord vil de udgøre en uafhængig delmængde i komplementet. Den anden påstand vises tilsvarende. ■

Lad os se på et andet mere interessant eksempel.

Eksempel 1.3.10. Vi betrager følgende graf G med syv knuder:



Vi ser, at knuderne A, G, F er parvist naboer, og derfor udgør de en klike. Det samme gælder A, B og C, D, E, F. Dermed har G en klike med både 2, 3 og 4 knuder. Vi ser desuden, at A, C er en uafhængig delmængde, og at B, D, G er en uafhængig delmængde. ∘

Når vi skal farvelægge grafer, bliver det vigtigt at se på størrelsen af kliker og uafhængige delmængder.

Definition 1.3.11. Lad G være en graf. *Kliketallet* for G, betegnet $\omega(G)$ er det maksimale antal knuder, der kan være i en klike i G. *Uafhængighedstallet* for G, betegnet $\alpha(G)$, er det maksimale antal knuder, der kan være i en uafhængig delmængde af G.

ω er et lille omega, og er det sidste bogstav i det græske alfabet. α er et lille alfa og er det første bogstav i det græske alfabet. Dette stemmer i en vis forstand overens med, at kliker og uafhængige delmængder er modsatte begreber.

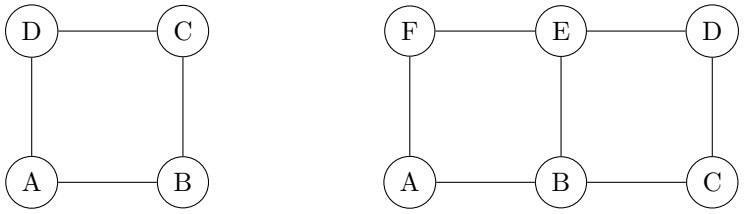
Eksempel 1.3.12. Lad G være grafen fra eksempel 1.3.10. Vi ser, at vi kan finde en klike af størrelse 4, nemlig C, D, E, F. Man kan også overbevise sig selv om, at der ikke findes en større klike i G, og derfor er $\omega(G) = 4$. I eksemplet fandt vi også en uafhængig delmængde med tre knuder, nemlig B, D, G, og denne samling af knuder har maksimal størrelse, så $\alpha(G) = 3$. Kan du finde en anden uafhængig delmængde med tre knuder? ∘

Todelte grafer

Todelte grafer er en type af grafer, som bliver interessant, når vi skal tale om farvelægning af grafer. I dette afsnit kommer vi også til at gennemgå nogle interessante resultater, som er sværere at bevise i forhold til de sætninger, vi har vist indtil nu.

Definition 1.3.13. En graf G kaldes *todelt*, hvis knuderne i G kan opdeles i to uafhængige delmængder, hvor ingen knuder i den ene uafhængige delmængde ligger i den anden. Sådanne to uafhængige delmængder kaldes en *todeling* eller en *bipartition*.

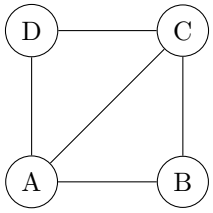
Eksempel 1.3.14. Betragt de to grafer herunder:



I grafen til venstre har vi A, C og B, D som uafhængige delmængder, og tilsammen udgør disse fire knuder alle knuder i grafen, så den venstre graf er todelt. For grafen til højre kan det ses, at A, E, C og B, D, F udgør en todeling. ∘

Lad os for god orden skyld også betragte et ikke-eksempel på en todelt graf. Med andre ord, et eksempel på en graf, der ikke er todelt.

Eksempel 1.3.15. Lad G være grafen herunder:



Vi hævder, at G ikke er todelt. Hvis vi har en todeling af grafen, skal A ligge i en af de uafhængige delmængder. Men A er nabo til alle andre knuder, så den ene uafhængige delmængde skal være A. Vi kan dog på samme måde sige, at C bliver nødt til at udgøre sin egen uafhængige delmængde, og dermed mangler vi en uafhængig delmængde med både B og D. Ergo kan en todeling ikke eksistere, så G er ej todelt. ∘

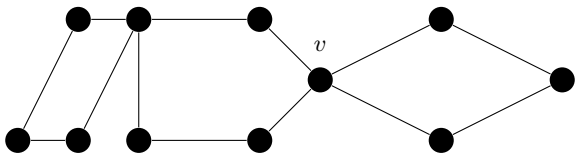
Det er ikke svært at forestille sig grafer, hvor det kan blive meget besværligt at afgøre, om den er todelt eller ej. Hvis vi kun har definitionen til rådighed, skal vi tjekke, at samtlige opdelinger ikke fungerer, og det kan hurtigt komme til at kræve meget tid og arbejde. For at lette vores arbejde med at bestemme, om en graf er todelt eller ej, vil vi bruge resten af dette afsnit på at formulere og bevise et kriterium for, om en graf er todelt. Beviserne er fra [4].

For at bevise følgende nyttige hjælperesultat bruger vi den bevisteknik, som kaldes *induktion*. Hvis man har en påstand, som involverer et positivt heltal n, kan man bevise, at påstanden gælder for alle n ved først at vise, at påstanden gælder for $n = 1$ (induktionsstarten), hvorefter man viser, at det gælder for $n > 1$ under antagelsen af, at det gælder for $n - 1$ (induktionsskridtet). Dette kaldes simpel induktion. Ud fra simpel induktion kan man vise *princippet om fuldstændig induktion*: i induktionsskridtet kan man faktisk antage, at det gælder for alle $k < n$. Husk, at længden af en rute er lig antallet af kanter i ruten.

Lemma 1.3.16. *Alle lukkede ruter af ulige længde indeholder en kreds af ulige længde.*

Bevis. Lad l betegne længden af vores rute. Vi benytter induktion på l. Hvis $l = 1$, må ruten selv udgøre en kreds af længde 1, som er ulige. Dette viser induktionsstarten. Lad nu $l > 1$. Antag først, at der ingen gentagne knuder er i ruten. Fordi ruten er lukket, vil den udgøre en kreds (af ulige længde), så vi er færdige. Antag nu, at der er en knude, v, som besøges to gange i løbet af ruten. Da kan den oprindelige rute opdeles i to ruter, der starter og slutter i v, og disse har begge længde mindre end l. Da l er ulige, skal den ene af ruterne have ulige længde, og induktionsantagelsen giver, at denne mindre rute har en kreds af ulige længde. Dette færdiggør beviset. ■

Vi kan illustrere beviset for lemmaet med følgende tegning:



Man kunne tro, at alle lukkede ruter af *lige* længde må indeholde en cykel af lige længde. Dette er dog ikke rigtigt, se opgave 1.3.22.

Lemma 1.3.17. *Lad G være en todelt graf. En todeling for G kan findes ved at finde en todeling af alle sammenhængskomponenter. Mere præcist: Hvis $H_1, ..., H_k$ er sammenhængskomponenterne i G, og vi har en todeling X_i, Y_i af hver H_i, da vil en todeling af G være givet ved at lægge alle $X_i$erne sammen og alle $Y_i$erne sammen.*

Bevis. Lad X være knuderne i $X_1, ..., X_k$ og Y knuderne i $Y_1, ..., Y_k$. Det er klart, at X og Y tilsammen udgør G, og at de ikke har nogle knuder til fælles. Vi mangler da kun at vise, at X og Y hver er uafhængige delmængder. Antag, at v_1 og v_2 er knuder i X, som er naboer. Da bliver v_1 og v_2 nødt til at ligge i samme sammenhængskomponent for G. Lad os sige, at denne er H_i. Eftersom v_1 og v_2 er naboer, skal vi have, at v_1 og v_2 ligger i hver deres del af todelingen for H_i. Vi kan antage per symmetri, at v_1 ligger i X_i, og at v_2 ligger i Y_i. Men da ligger v_2 i Y, da Y jo indeholder knuderne i Y_i. Men dette er en selvmodsigelse, eftersom v_2 ligger i X per antagelse, og X og Y ingen knuder har tilfælles. Dermed kan ingen knuder i X være naboer, så X er en uafhængig delmængde. Et fuldstændig analogt argument viser, at Y er en uafhængig delmængde. Dette fuldfører beviset. ■

Nu kan vi vise dette afsnits hovedresultat.

Sætning 1.3.18 (**Königs sætning**). *En graf er todelt hvis og kun hvis den ikke har en kreds af ulige længde.*

Bevis. Antag først, at G er en todelt graf. Alle ruter i G hopper skiftevis frem og tilbage mellem de to dele af todelingen. Det følger, at alle kredse bliver nødt til at have lige længde, eftersom en kreds starter i en af de to dele og skal vende tilbage til samme del. Altså har G ingen kredse af ulige længde. Antag nu, at G ingen kredse har af ulige længde. Per forrige lemma er det tilstrækkeligt at vise, at alle sammenhængskomponenter har en todeling, så lad H være en ikke-triviel (dvs. H har mindst én kant) sammenhængskomponent for G (hvis G ikke har sådan en sammenhængskomponent, har G ingen kanter, og en todeling findes oplagt). Lad u være en knude i H. Hvis v er en anden knude i H, lad da $f(v)$ betegne den mindste længde til knuden v fra u. Da H er sammenhængende, eksisterer tallet $f(v)$ for alle v i H. Lad nu X være de knuder i H, hvor $f(v)$ er lige, og Y være de knuder i H, hvor $f(v)$ er ulige. Det er klart, at X og Y ikke har nogle knuder til fælles (et tal er ikke både lige og ulige), og at X og Y tilsammen udgør H. Hvis der var to knuder v_1 og v_2 i X, som var naboer, da ville vi kunne lave en lukket rute fra u til v_1 til v_2 tilbage til u. Bemærk, at denne rute har ulige længde (1 + lige + lige = ulige). Per lemma 1.3.16 vil denne lukkede rute have en kreds af ulige længde, hvilket er i modstrid med antagelsen om G. Derfor er ingen knuder i X naboer, så X må være en uafhængig delmængde. Beviset for, at Y er en uafhængig delmængde, forløber tilsvarende. Den lukkede rute vil være ulige, fordi summen af to ulige tal er lige, så man opnår den samme modstrid. ■

Sætningen ovenover blev bevist af Dénes König og er derfor opkaldt efter ham. König udgav den første bog om grafteori i 1936 ved navn *Theorie der endlichen und unendlichen Grafen* (på dansk: Teorien om endelige og uendelige grafer) [5].

Opgaver til Flere begreber og egenskaber for grafer

Vi konkluderede i eksempel 1.3.2, at alle kredsgrafer har lukkede Euler-ture. I de næste tre opgaver ser vi på de to andre typer af standardgrafer K_n (komplet graf) og P_n (vejgraf) og undersøger, om disse har Euler-ture (åbne eller lukkede).

- **Opgave 1.3.1:**
  [Vink: Tegn graferne!]

  1) Har P_4 en Euler-tur? Hvis ja, er den så åben eller lukket?

  2) Har P_5 en Euler-tur? Hvis ja, er den så åben eller lukket?

  3) Har P_6 en Euler-tur? Hvis ja, er den så åben eller lukket?

  4) Har en vilkårlig vejgraf P_n en Euler-tur? Hvis ja, er den så åben eller lukket?

- **Opgave 1.3.2:**
  [Vink: Tegn graferne!]

  1) Har K_4 en Euler-tur? Hvis ja, er den så åben eller lukket?

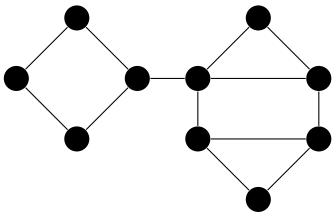
  2) Har K_5 en Euler-tur? Hvis ja, er den så åben eller lukket?

  3) Har K_6 en Euler-tur? Hvis ja, er den så åben eller lukket?

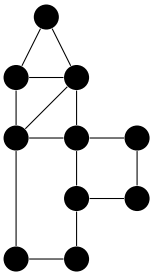
  4) Hvilke komplette grafer har en Euler-tur? Hvad skal der gælder om n?

- **Opgave 1.3.3:**
  Denne opgave er fra [2]. Afgør om graferne har en lukket eller åben Euler-tur, og find om muligt en.

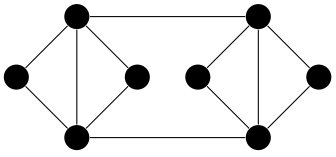
  1)



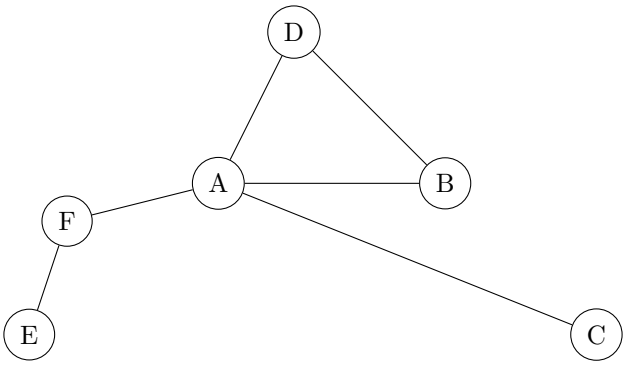
  2)



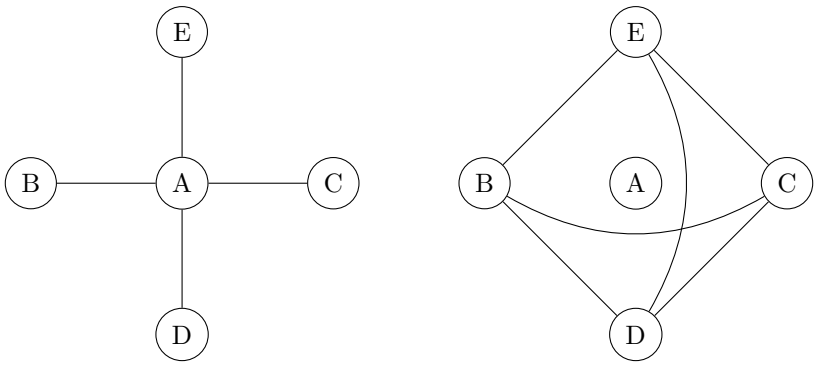
  3)



- **Opgave 1.3.4:**
  Tegn komplementet til grafen herunder:



- **Opgave 1.3.5:**
  Tegn komplementet til hver af de to grafer herunder:



- **Opgave 1.3.6:**
  Lad G være en simpel graf med n knuder. Vis, at $G = K_n$, hvis og kun hvis $\overline{G}$ ingen kanter har. [Husk, at "hvis og kun hvis" betyder "er det samme som".]

- **Opgave 1.3.7:**
  Lad G være grafen i eksempel 1.3.10. Tegn komplementet for G og vis, at de fundne kliker i eksemplet bliver til uafhængige delmængder i $\overline{G}$.

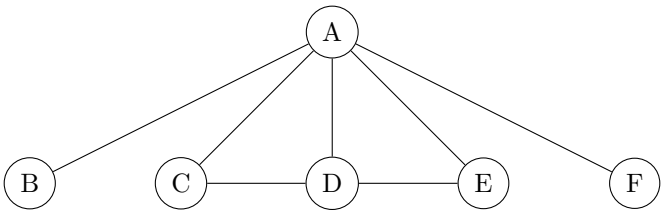
- **Opgave 1.3.8:**
  Bestem kliketallet og uafhængighedstallet for grafen i opgave 1.3.4.

- **Opgave 1.3.9:**
  Bestem kliketallet og uafhængighedstallet for graferne i opgave 1.3.5.

- **Opgave 1.3.10:**
  Hvad er kliketallet og uafhængighedstallet for den komplette graf med n knuder, K_n?

- **Opgave 1.3.11:**
  Lad G være en simpel graf. Vis, at $\alpha(G) = \omega(\overline{G})$ og $\alpha(\overline{G}) = \omega(G)$. Med andre ord: uafhængighedstallet af G er lig kliketallet for $\overline{G}$ og omvendt.

• **Opgave 1.3.12:**
Bestem kliketallet og uafhængighedstallet for grafen G tegnet herunder:



•• **Opgave 1.3.13:**
Hvad er kliketallet og uafhængighedstallet for vejgrafen med n knuder, P_n? [Vink: Kig på tilfældende, hvor der er 1, 2, 3, 4 og 5 knuder og se, om du kan gennemskue et mønster.]

••• **Opgave 1.3.14:**
I denne opgave vil vi bestemme kliketallet og uafhængighedstallet for kredsgraferne.

1) Udfyld tabellen herunder:

n	1	2	3	4	5	6	7	8
$\omega(C_n)$								
$\alpha(C_n)$								

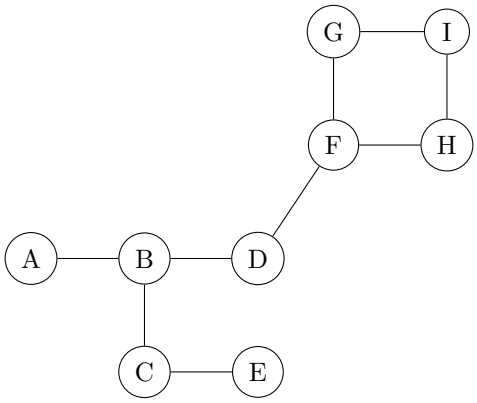
2) Baseret på den tabel, du udfyldte i forrige opgaver, kan du forklare mønsteret i kliketallet og uafhængighedstallet? Kan du give en eksplicit formel (muligvis for tilpas store n)?

•• **Opgave 1.3.15:**
Brug Königs sætning til at give et nemmere bevis for, at grafen i eksempel 1.3.15 ikke er todelt.

• **Opgave 1.3.16:**
Er grafen i eksempel 1.3.10 todelt? Hvad med grafen i opgave 1.3.12?

•• **Opgave 1.3.17:**
Forklar, hvorfor at alle træer er todelte. Kan du give en fremgangsmåde til at finde en todeling af et træ med n knuder? Se opgave 1.2.12. Hvis du ikke har lavet opgaven om træer, kan du springe denne opgave over.

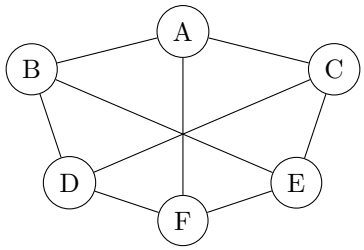
•• **Opgave 1.3.18:**
Betragt grafen G herunder:



1) Vis, at grafen er todelt ved at bruge Königs sætning.

2) Find en todeling af grafen.

●● **Opgave 1.3.19:**
Betragt grafen G herunder:



1) Vis, at grafen er todelt ved at bruge Kónigs sætning.

2) Find en todeling af grafen.
Grafen G er et eksempel på en såkaldt *Haar*-graf, og den betegnes $H(7)$.

●● **Opgave 1.3.20:**
Vi skal i denne opgave undersøge, hvilke komplette grafer, der er todelte. Som sædvanligt er Königs sætning et godt sted at starte.

1) Er K_2 todelt?

2) Er K_n todelt, når n er større end 2?

●● **Opgave 1.3.21:**
I denne opgave undersøger vi, hvornår kredsgraferne er todelte.

1) Er C_2, C_4 og C_6 todelte?

2) Er C_3, C_5 og C_7 todelte?

3) For hvilke værdier af n er C_n todelt?

●● **Opgave 1.3.22:**
Giv et eksempel på en lukket rute af lige længde, som ikke har en cykel af lige længde. Konkludér, at lemma 1.3.16 kun gælder for lukkede ruter af ulige længde.

●●● **Opgave 1.3.23:**
I denne opgave viser vi, at for en todelt graf G gælder det, at en todeling er unik hvis og kun hvis G er sammenhængende.

1) Antag, at G *ikke* er sammenhængende, dvs. der er mindst to sammenhængskomponenter. Vis, at der findes mindst to forskellige måder at lave en todeling på. [Vink: lemma 1.3.17, læs evt. beviset for at få den rigtige idé].

Den første opgave viser, at hvis G har en unik todeling, så er G sammenhængende (vi kalder denne bevisstrategi for *kontraposition*). Antag, nu at G er sammenhængende og lad X, Y være en todeling af G. I de følgende delopgaver viser vi, at todelingen X, Y er unik. Lad v være en knude i X.

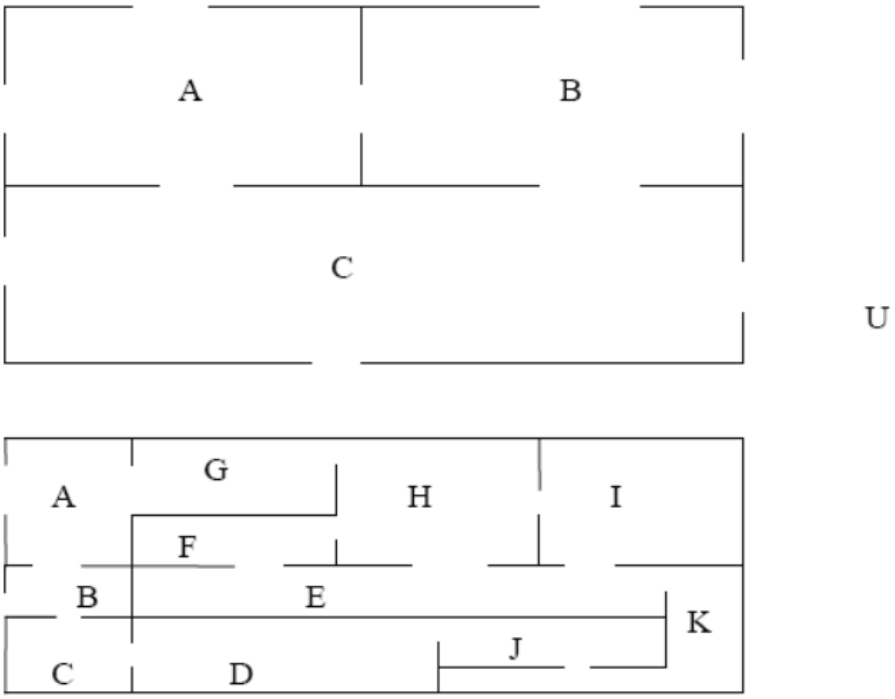
2) Lad X', Y' være en todeling af G. Vi kan per symmetri antage, at v ligger i X'. Vi er da færdig, idet vi har vist, at $X' = X$. Hvorfor?

3) For at vise, at $X' = X$, skal vi vise, at for en knude v' i X' skal v' også ligge i X. Antag for modstrid, at v' ligger i Y. Forklar, hvorfor der findes en rute fra v til v'.

4) Hvorfor har ruten fra v til v' ulige længde? [Vink: læs første del af beviset for Königs sætning.]

5) Brug forrige delopgave til at vise, at v' ligger i Y'. Forklar hvorfor, at dette er en modstrid og konkludér, at v' ligger i X.

●● **Opgave 1.3.24:**
Denne opgave er fra [2]. Herunder er tegnet en plan over to huse. Tegn passende grafer over begge huse, så rummene er knuder og dørene kanter (betragt haven U som et rum i hvert hus). Afgør herefter, om det er muligt at gå en tur gennem huset, så man går gennem hver dør netop én gang.



Figur 1.9: Plan over de to huse.

●● **Opgave 1.3.25:**
Kan man finde en lukket Euler-tur over broerne i Königsberg?

●● **Opgave 1.3.26:**
Kan man finde en åben Euler-tur over broerne i Königsberg?

●●● **Opgave 1.3.27:**
Lad G være en graf med en lukket Eulertur.

1) Bevis, at G er sammenhængende.

2) Bevis, at alle knuder i G har lige valens.

●●● **Opgave 1.3.28:**
Lad G være en graf med en åben Eulertur.

1) Bevis, at G er sammenhængende.

2) Bevis, at netop to knuder i G har ulige valens. [Vink: hvad skal gælde om valensen af de knuder, som turen starter og slutter i? Hvad med alle andre knuder?]

1.4 Farvelægning af grafer

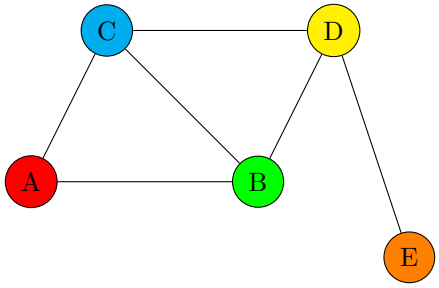
Hvad er en farvelægning?

Vi antager fra nu af, at alle grafer er sammenhængende.

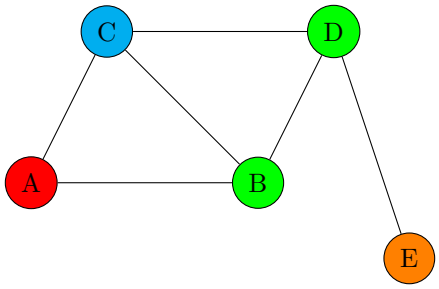
I dette kapitel vil vi arbejde med farvelægning af grafer. I har muligvis hørt om fire-farvesætningen, der siger, at givet et (relativt pænt) kort med forskellige lande, er det altid muligt med fire farver at farvelægge landene således, at ingen nabolande har samme farve. Dette inspirerer følgende definition.

Definition 1.4.1. En farvelægning af en graf er et valg af farve til enhver kunde, således at ingen naboer har samme farve.

Følgende er en farvelægning.



Det er dog en meget kedelig farvelægning; alle farverne er nemlig forskellige! Følgende er **ikke et eksempel på en farvelægning**,



da B og D har samme farve, men også er naboer. Vi kan helt klart slippe afsted med færre end fem farver til den ovenstående graf. Men fem er i hvertfald muligt. Dette motiverer følgende definition.

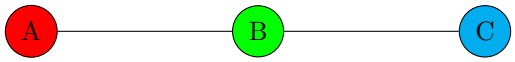
Definition 1.4.2. Hvis vi kan finde en farvelægning af en graf, der kun kræver k forskellige farver, siger vi, at den kan k-farves.

Hvis en graf G har n knuder, kan vi altid farvelægge den med $k = n$ forskellige farver: én per knude! Dog kan man ofte slippe afsted med færre:

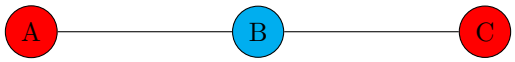
Definition 1.4.3. For en graf G, definerer vi $\chi(G)$ til at være det mindste tal k, så G kan k-farves. $\chi(G)$ kaldes grafen G's *kromatiske tal*.

Betragt følgende farvelægning af en graf

Denne farvelægning viser, at G har et kromatisk tal på 2, for vi kan ikke farve den med 1 farve, men 2 er nok. Betragt nu følgende farvelægning af en graf:



Dette viser at at det kromatiske tal for denne graf højest kan være 3, men vi ved ikke om det kunne være lavere. Det kan det faktisk:



Dette er også en farvelægning, så det kromatiske tal er 2. En god taktik til at finde kromatiske tal er følgende

1. Find en farvelægning, der bruger k farver.
2. Vis, at der ikke findes nogen n-farvelægning for $n < k$.

Dog kan dette blive besværligt, så vi har brug for nogle værktøjer.

Nedre grænser

Hvordan relaterer kliketallet $\omega(G)$ sig til det kromatiske tal $\chi(G)$? Vi har følgende resultat:

Sætning 1.4.4. *For en graf G er det kromatiske tal højere end (eller lig med) kliketallet, det vil sige*

$$\omega(G) \leq \chi(G) \tag{1.1}$$

Bevis. Lad $v_1, \ldots, v_n$ være en samling knuder, som udgør en maksimal klike, det vil sige $n = \omega(G)$. Hvis vi nu farvelægger grafen G, kan vi sige følgende om farverne:

- v_2 er nabo med v_1, så v_2 har ikke samme farve som v_1. Så vi har brug for mindst 2 forskellige farver.
- v_3 er nabo med v_1 og v_2, så v_3 har ikke samme farve som hverken v_1 eller v_2. Så vi har brug for mindst 3 forskellige farver.
- v_4 er nabo med v_1, v_2 og v_3, så v_4 har ikke samme farve som hverken v_1, v_2 eller v_3. Så vi har brug for mindst 4 forskellige farver.
- Og så videre og så videre, indtil vi efter $n-1$ skridt når til v_n og konkluderer, at vi har brug for mindst n farver.

Sagt med andre ord har vi, at $\omega(G) = n \leq \chi(G)$ ■

Vi kan ligeledes relatere uafhængighedstallet til det kromatiske tal.

Sætning 1.4.5. *For en graf G med n knuder har vi*

$$\frac{n}{\alpha(G)} \leq \chi(G)$$

Bevis. Farvelæg først G med $k = \chi(G)$ forskellige farver. Hvis vi nummerer farverne med $1, 2, 3, \ldots, k$, kan vi samle alle knuder farvet med farve 1 i en bunke F_1, farve 2 i F_2 og så videre. Alle knuder er i *præcis* én af de her bunker, så

$$n = |F_1| + |F_2| + \cdots + |F_k|$$

hvor $|F_i|$ er antallet af knuder i F_i. $|F_i|$ er altid mindre end eller lig med $\alpha(G)$, så

$$\begin{aligned} |F_1| + |F_2| + \cdots + |F_k| &\leq \overbrace{\alpha(G) + \alpha(G) + \cdots + \alpha(G)}^{k \text{ gange}} \\ &= k\alpha(G) = \chi(G)\alpha(G) \end{aligned}$$

Altså har vi, at $\frac{n}{\alpha(G)} \leq \chi(G)$. ■

Den grådige algoritme og en øvre grænse

Vi har indtil videre kun set eksempler på størrelser, der giver en nedre grænse for det kromatiske tal $\chi(G)$ for en graf. Men det kunne være rart at have en øvre grænse. Vi vil dog først gennemgå en algoritme til at farvelægge en graf.

Den grådige algoritme

For en graf G, vælg en nummerering af alle knuderne i grafen $v_1, \ldots, v_n$ og vælg ydermere n forskellige farver, og nummerer dem fra 1 til n. Gør nu følgende:

- Farvelæg v_1 med farve nr. 1.
- Gør nu følgende for knude nr. i, v_i:
  - Hvis den ikke har nogen nabo med farve nr. 1, så giv den farve nr. 1. Ellers forsæt til næste skridt.
  - Hvis den ikke har nogen nabo med farve nr. 2, så giv den farve nr. 2. Ellers forsæt til næste skridt.
  - Dette mønster forsættes indtil knuden er farvelagt.

Denne algoritme er afhængig af en nummerering. Hvis vi skal håbe på at få en nogenlunde fornuftig farvelægning med den, så har vi brug for en fornuftig måde at nummerere dem.

Definition 1.4.6. Lad v og u være to knuder i en graf G. Hvis de er forbundne, så lader vi $d(v, u)$ være antallet af kanter i den korteste rute fra v til u. Dette er *afstanden* mellem v og u.

Sætning 1.4.7. *Lad $\Delta(G)$ være den maksimalvalensen for G. Da gælder det, at*

$$\chi(G) \leq \Delta(G) + 1$$

Bevis. Lad $n = n(G)$ være antallet af knuder i G. Vælg en knude v_n, og sortér nu alle knuder $v_1, \ldots, v_n$ efter deres afstand til v_n, så $d(v_i, v_n) \geq d(v_j, v_n)$, når $i < j$. Med andre ord kommer knuderne, der er længst væk fra v_n, først i listen. Vi påstår nu, at hvis vi anvender den grådig algoritme på denne nummerering, så benyttes ikke mere end $\Delta(G) + 1$ farver.

v_1 er farvelagt med farve 1, så ingen problemer der. Lad os nu se, hvad der sker, når vi skal til at farvelægge knude nr. i, hvor $1 < i < n$. v_i har mindst én nabo, der endnu ikke er farvelagt: Vælg blot en knude på den korteste rute fra v_i til v_n. Knuderne på denne rute er ikke farvelagte endnu, da de alle har en mindre afstand til v_n end v_i har. Men v_i har allerhøjest $\Delta(G)$ naboer, hvoraf højest $\Delta(G) - 1$ er farvelagte. Så selv i det værste tilfælde bruger algoritmen farve nr. $\Delta(G)$.

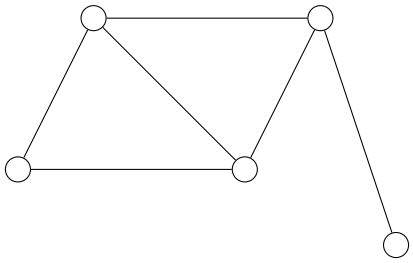
Hvad så med v_n? Den har højest $\Delta(G)$ naboer, så i værste fald bruger den grådige algoritme farve nr. $\Delta(G) + 1$.

Vi kan altså konkludere, at $\chi(G) \leq \Delta(G) + 1$. ■

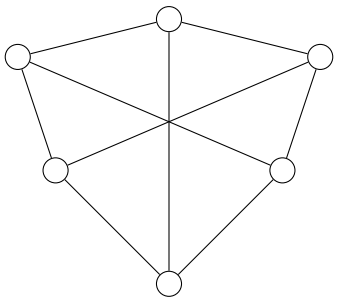
Opgaver til Farvelægning af grafer

- **Opgave 1.4.1:**
  Kan alle grafer farvelægges?

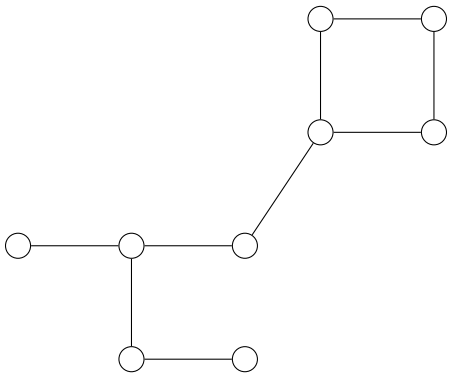
- **Opgave 1.4.2:**
  Find det mindste k, så følgende graf kan k-farves, dvs. en optimal farvelægning med $\chi(G)$ farver.



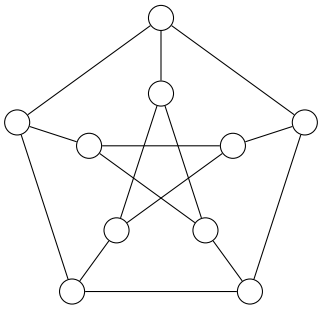
- **Opgave 1.4.3:**
  Find det mindste k så følgende graf kan k-farves, dvs. en optimal farvelægning med $\chi(G)$ farver.



- **Opgave 1.4.4:**
  Find det mindste k så følgende graf kan k-farves, dvs. en optimal farvelægning med $\chi(G)$ farver.



- **Opgave 1.4.5:**
  Find det mindste k så følgende graf kan k-farves, dvs. en optimal farvelægning med $\chi(G)$ farver.



●● **Opgave 1.4.6:**
I denne opgave undersøger vi grænsen $\omega(G) \leq \chi(G)$:
1) Findes der en graf G hvor $\omega(G) = \chi(G)$?
2) Findes der en graf G hvor $\omega(G) < \chi(G)$?

●● **Opgave 1.4.7:**
I denne opgave undersøger vi grænsen $n(G)/\alpha(G) \leq \chi(G)$:
1) Findes der en graf G hvor $n(G)/\alpha(G) = \chi(G)$?
2) Findes der en graf G hvor $n(G)/\alpha(G) < \chi(G)$?

●● **Opgave 1.4.8:**
I denne opgave undersøger vi grænsen $\chi(G) \leq \Delta(G) + 1$:
1) Findes der en graf G hvor $\chi(G) = \Delta(G) + 1$?
2) Findes der en graf G hvor $\chi(G) < \Delta(G) + 1$?

●● **Opgave 1.4.9:**
Lad G være en graf. Antag, at maksimalvalensen er mindre end eller lig med to.
1) Hvad kan man sige om $\chi(G)$? Hvor stor og hvor lille kan den blive?
2) Findes der grafer, der opnår de grænser du har fundet for $\chi(G)$?

●● **Opgave 1.4.10:**
Lad G være en graf. Antag, at maksimalvalensen er mindre end eller lig med fem.
1) Hvad kan man sige om $\chi(G)$? Hvor stor og hvor lille kan den blive?
2) Findes der grafer, der opnår de grænser du har fundet for $\chi(G)$?

●● **Opgave 1.4.11:**
I definition 1.4 er den grådige algoritme beskrevet. Finder den altid den optimale farvelægning? [Vink: Prøv at bruge den grådige algoritme på nogle grafer.]

●● **Opgave 1.4.12:**
Finder den grådige algoritme nogensinde en optimal farvelægning. [Vink: Prøv at bruge den grådige algoritme på nogle grafer.]

●●● **Opgave 1.4.13:**
Er der nogen grænse for hvor mange farver den grådige algoritme bruger udover dem der er krævet til en optimal farvelægning? [Vink: Du kan bruge de forrige opgaver her.]

● **Opgave 1.4.14:**
Find på nogle grafer der kan 2-farves.

●● **Opgave 1.4.15:**
Find på nogen betingelser for at en graf kan 2-farves. [Vink: Tag udgangspunkt i den forrige opgave.]

●● **Opgave 1.4.16:**
Find nogen betingelser, som er nødvendige for at kunne 2-farves. Dvs. betingelser som hvis de ikke er opfyldt garantere, at grafen ikke kan 2-farves.

●●● **Opgave 1.4.17:**
Find på en graf, med højst 10 kanter, som kræver mange farver at farve.

Kapitel 2

Fysik

Dette års forløb på fysik vil fokusere på naturvidenskabens eksperimentelle karakter. Forhåbentlig vil du få muligheden for at reflektere over hvordan man sætter et godt eksperiment op, tager data systematisk og behandler data på en fornuftig måde. Det er færdigheder der kan bruges i de fleste af de andre fag repræsenteret i kompendiet, og naturligvis i livet og din fremtidige uddannelse generelt!

I folkeskolen følger eksperimenter ofte et kedeligt mønster: Man får udleveret en vejledning der bare skal følges, og får ikke ret megen mulighed for selv at overveje hvorfor man gør som man gør. I dette forløb vil du i stedet blive bedt om selv at designe alt selv.

Desværre er det ikke ret let for en folkeskoleelev at undersøge fysikkens nuværende fronter, da det ofte kræver stort og dyrt udstyr, som teleskoper, satellitter eller partikelacceleratorer. Derfor vil vi i stedet stille os i tideligere tiders fysikeres sko, og undersøge et objekt som enhver kan fremstille. Vi skal tilbage til 1600-tallet hvor fysikere som Galileo og Huygens undersøgte de matematiske love som svingningen af et pendul følger. De levede i en tid før den klassiske mekanik var færdigudviklet, og havde derfor omtrent de samme forudsætninger for at undersøge et pendul som en moderne folkeskoleelev. [1] For ikke at friste til at slå svarene op, er al teori og facit bevidst udeladt fra kompendiet - Galileo og Huygens havde heller ikke an facitliste! Derudover vil alle opgaver i forløbet kunne løses med papir, blyant, lineal, stopur og den menneskelige hjerne. [2]

I princippet kunne vi stoppe fysikdelen af kompendiet her, og overlade det til dig at regne resten ud selv - det kan du sagtens, men for dem der gerne vil have lidt hjælp eller inspiration vil vi i resten af afsnittet gennemgå den naturvidenskabelige metode, god eksperimentel praksis, plot af data, funktionsbegrebet og fit til data.

2.1 Naturvidenskabelig metode

Naturvidenskab er bygget på eksperimenter. Måden vi finder ud af hvordan verden fungerer er kort sagt ved at prøve en masse forskellige eksperimenter og vigtigst af alt; skrive det vi ser ned. Det er derfor værd at spørge hvordan man egentlig laver et godt eksperiment? Og hvordan man sikrer sig at man bruger den data man får ordentligt og ikke drager forkerte konklusioner fra den. Det er det vi skal finde ud af her.

[1] På nær at de skrev deres opdagelser ned på latin...

[2] Hvis du ikke gidder hovedregning er det okay at bruge en lommeregner

Den naturvidenskabelige metode bygger på at man starter med en hypotese, som man kan teste ved et forsøg.[3] Ved at behandle de opsamlede data kan man så enten bekræfte eller afkræfte hypotesen og eventuelt lave en ny hypotese ud fra resultaterne og starte forfra. Lad os konkretisere med et kort eksempel:

- Hypotese: Hvordan tror du verden fungerer?
  - Isabella cykler hurtigere end Kasper.[4]
- Forsøg: Afprøv om din forudsigelse holder, ved at prøve det og skrive resultaterne ned.
  - Lad Isabella og Kasper cykle den samme distance flere gange og tag tid.
- Databehandling: Sammenlign dine data på meningsfuld vis.
  - Tag gennemsnit af Isabellas tider og Kaspers tider og sammenlign dem.
- Be- eller afkræft hypotesen: Havde du ret?
  - Var Isabella eller Kasper hurtigst og var der stor nok forskel på tiderne til at du er sikker på at den ene er hurtigere end den anden?

Helt centralt i den naturvidenskabelige metode er hypotesen. Den er vores udgangspunkt for at skabe vores viden og derfor er det vigtigt at den er god, men hvad vil det overhovedet sige at en hypotese er god? Man plejer at sige at en god hypotese er konkret og testbar. For eksempel er hypotesen "Sten er store" ikke specielt god da det er uklart hvad der menes med at en sten er stor (snakker vi på størrelse med et hønseæg eller med en personbil?) og derudover er det heller ikke helt klart hvilken definition af ordet sten vi bruger. Den skal også være testbar det vil sige at hypotesen "Der bor usynlige feer på mit loft, der ikke siger en lyd, ikke kan mærkes og ikke vejer noget" er en dårlig hypotese da vi ikke har nogen måde at detektere feerne på og den dermed ikke kan testes.

Design af eksperimenter

Hvis man prøver at undersøge om der er forskel på hvem af dine venner der er hurtigst på cykel, er det så *fair*, hvis den ene skal cykle opad bakke og den anden nedad bakke? Den ene i regnvejr den anden i tørke?

For at finde ud af hvem der er hurtigst, skal man jo måle hvor lang tid det tager dem at cykle en bestemt afstand. Det kan man gøre på mange måder. Man kan stå klar med et stopur og starte og stoppe det som de starter og stopper cykelturen. Et billigere alternativ er at tælle sekunderne der går fra processen begynder til den slutter. Men hvilken måde er *bedst* og hvorfor? Hvis nu de to venner cykler næsten helt lige hurtigt, så betyder det noget, hvor sikre vi er på den tid det tog dem. F.eks. hvis begge cykelture tog hhv. 58s og 60s, hvor vi bare har talt i hovedet så vi har en ret stor usikkerhed på vores måling lad os gætte på ca. $\pm$3s, altså det kan altså godt være at det tog 3 sekunder mere eller mindre end det vi har målt det til. Jamen så kan cykelturen på 58s faktisk godt have været på 61s og omvendt den på 60s kunne godt have taget 57s, så den hurtigste cykeltur måske var den langsommeste, vi kan ikke være sikre på hvem der er hurtigst!

Variabelkontrol

Når vi skal finde ud af om en bestemt ændring påvirker forsøget (og hvordan) f.eks. om det betyder noget om det er Birgitte eller Peter der cykler, for hvor lang tid cykelturen tager, så skal vi være fair mod begge to. De skal testes under de samme forhold. Det ville

[3] At hypotesen er testbar er meget vigtigt! Helst skal den faktisk også være falsificerbar, det vil sige at den skal kunne modbevises.

[4] Bemærk at dette kan modbevises, hvis Kasper cykler hurtigere

ikke være *fair* at konkludere at når Birgitte cykler så går det dobbelt så hurtigt som når Peter gør det, hvis hun cyklede i medvind og solskin imens Peter cyklede i stormvejr og modvind. Når vi gerne vil se hvordan ændringer af en variabel påvirker forsøget så er vi nødt til at holde alt andet, så godt som muligt, konstant. F.eks. hvordan ændringen af cykelrytter påvirker cykelturens tid. Vi kan jo ikke være sikrer på at det er pga. cykelrytteren at turen tog længere eller kortere tid, hvis de skulle cykle under meget forskellige forhold.

Dette er hvad der kaldes variabelkontrol. Altså når vi tester hvordan ændringen af én variabel påvirker forsøget skal vi holde alle andre konstante.

Et godt forsøg sørger for at de målinger der tages har en lav usikkerhed og sørger for variabelkontrol for alle variable de tester.

Lad os nu se på et par eksempler på et dårligt designet forsøg og et godt designet forsøg.

Et tvivlsomt eksperiment: Hoppehøjde af bolde

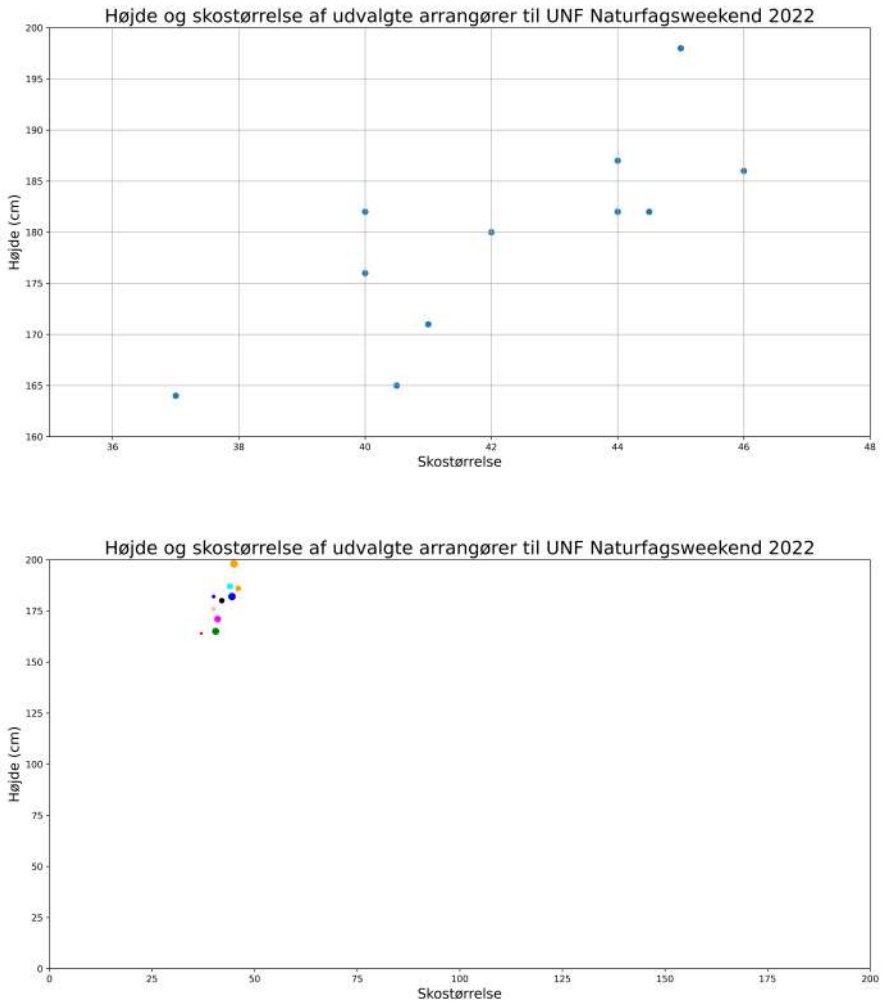
Vi vil gerne teste hvor højt bolde hopper når de kastes fra forskellige højder. Så vi tager og samler nogle forskellige bolde og taber dem fra forskellige højder og noterer hver gang hvor højt op de kom.

Op til flere ting kan være gået galt her. Hvis vi nu både har tabt fodbolde og hoppebolde, men ikke har noteret hvilke bolde der blev kastet hvornår vil vores resultater højest sandsynligt se meget mærkelige ud da en fodbold nok vil hoppe meget kortere end en hoppebold når den bliver tabt fra samme høje. Der er altså ikke blevet holdt styr på variablen boldtype. Man kunne også forestille sig at det ville gøre en forskel om man tabte bolden på græs eller asfalt, så underlag er en variabel man bør holde styr på.

Et godt eksperiment: EL-Effektivitet

Vi vil gerne vide hvilken måde at varme vand der er mest effektiv, vi undersøger: åben el-kedel; lukket el-kedel; forvarmet ovn; mikrobølgeovn. Vi tester alle tilfælde med 1L vand fra vandhanen. Vi måler effekten af alle redskaber imens, med den samme effektmåler. Alt vandet hældes ned i en beholder, hvor vi måler start-temperaturen. Derefter hælder vi 1L op i en el-kedel med åbent låg og varmer det i 5 min og tager straks slut-temperaturen af vandet, med samme termometer, vi finder også gennemsnits-effekten under opvarmningen. Vi hælder vadet fra el-kedelen ud (Vi ved nemlig ikke om det påvirker at vand er varmet en gang før så det varmes nemmere og så holder vi en fast starttemperatur). Vi lader el-kedlen køle af og gentager eksperimentet med nyt vand og lukket låg, i lige så lang tid og måler slut temperaturen og finder gennemsnitseffekten. Vi forvarmer en ovn og derefter slår vi effektmåleren til, så fylder vi en glas-beholder med 1L vand og sætter den ind i ovnen i de samme 5 min og måler slut temperaturen og gennemsnitseffekt. Det samme gør vi med mikrobølgeovnen.

Så kan vi finde varmeenergien der er tilføjet ift. hvor meget elektrisk energi der er blevet brugt. Et lille fejlkilde er at glasbeholderen måske kan påvirke forsøget ved at påvirke varmekapaciteten.

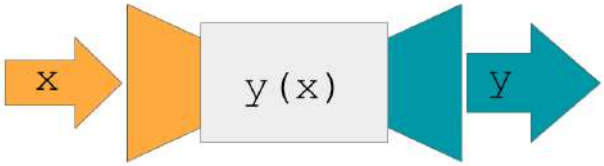


Figur 2.1: Et eksempel på et simpelt datasæt. Top: Plottet på en fornuftig måde. Bund: Plottet på en dårlig måde.

2.2 Plot af data

I naturvidenskab måler vi ofte to egenskaber ved samme objekt, og forsøger at undersøge om der er en sammenhæng mellem de to, og i så fald hvilken sammenhæng der er. Eksempler kunne være personers skostørelser og højder, et penduls længde og dets svingningstid eller elevers lommepenge og karakterer. Når vi har en mængde data, for eksempel kunne vi have spurgt ti mennesker om deres skostørrelse og højde, vil vi som det første *plotte* dataen. Det vil sige at vi indtegner et punkt i et koordinatsystem for hver person vi har spurgt, med skostørrelsen for den person som x-koordinat og højden af personen som y-koordinat. Se figur 2.1.

Et billede siger mere end tusind ord og de perfekte plots fortæller alt om deres eksperimenter. Helt ideelt så skal plots indholde information der forklarer hvad dataene i plottet er og hvad det data viser. Derfor er det noget af det vigtigste at skrive aksetitler og bruge ordentlige skalaer på de akser. Aksetitlerne kan forklare hvad der er henad den akse. Så er det også en god ide at bruge gode skalaer til at plotte sine data i. f.eks. et plot af menneske højder hvor vi plotter fra 0 til 10 meter har en masse tom plads som gør det svært at se de relative størrelser. Hvis man i stedet plotter fra 0 til vores højeste datapunkt. I figur 2.1 er der eksempler på to måder at plotte den samme data. I det nederste plot er det svært at se hvad der foregår fordi intervallet på akserne er valgt alt



Figur 2.2: Funktionsmaskinen, $f(x)$, tager imod et input, x og giver et output, y

for stort, og fordi punkter selv har forskellig størrelse og farve - selvom de er fra samme datasæt.

2.3 Funktionsbegrebet

En funktion oversætter noget den får til noget andet. f.eks. kan en funktion oversætte hvor lang tid du har cyklet til hvor langt du har cyklet. Man kan se på en funktion som en slags maskine, hvor du giver den et input og så giver den dig et output. Se figur 2.2. Vi plejer at skrive det som $f(x)$, det er en funktion af x. Hvis det nu var en funktion der forklarede afstand som funktion af tid så skriver vi ofte som $x(t) = v \cdot t$, afstanden, x, som funktion af tiden, t (hvor v beskriver en hastighed). Den kan vi så 'evaluere' i tidspunktet $t = 2s$. Det skriver vi (på lidt lang form) som $x(t = 2s) = v \cdot 2s$.

2.4 Funktionseksempler

Når vi har et plot af data, er det første spørgsmål man stiller; ligner det at der er en generel sammenhæng mellem de to målinger? I eksemplet med skostørrelserne ser det for eksempel ud til at større skosørrelse hænger sammen med større højde. Det næste spørgsmål er ofte 'kan vi gætte en funktion, der nogenlunde beskriver punkterne?'. Her er det enormt nyttigt at kende til graferne for et par almindelige funktioner, da man så nemt kan genkende dem i 'naturen'. Det er noget der mest af alt kommer med erfaring, og efterhånden som man møder de forskellige funktioner i sin matematikundervisning. For at give alle en basal fornemmelse af hvordan forskellige typer af funktioner kan se ud, er der tegnet nogle eksempler på funktionsgrafer i figur 2.3.

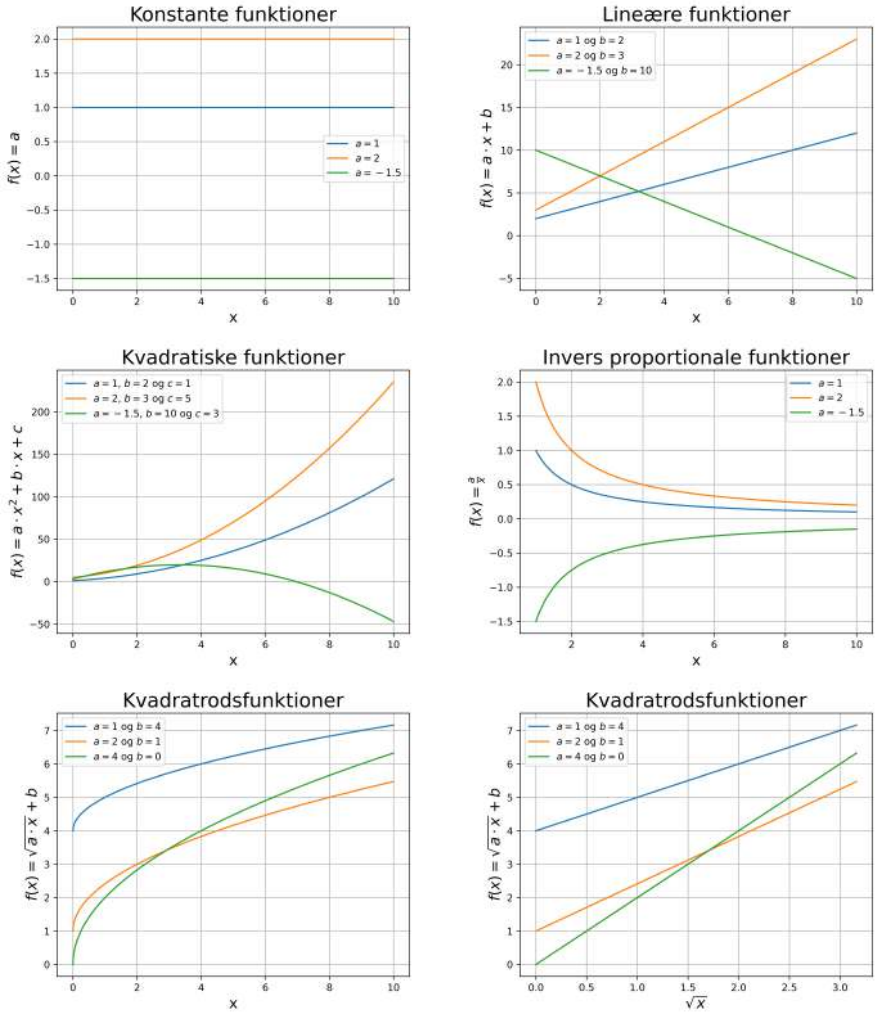
2.5 Lineære fits

Ofte er situationen at man har en mængde data, for eksempel skostørrelsen og højden af en række personer, og man nu ønsker at undersøge deres sammenhæng. Som nævnt tideligere er første skridt altid at plotte data. Ofte er vi i den situation at sammenhængen er lineær, og hvis den ikke er, vil vi senere komme ind på hvordan man kan transformere sine data således at de er lineære når man plotter dem.

Vi er altså interesseret i at finde parametrene a og b i den følgende sammenhæng:

$$f(x) = a \cdot x + b \tag{2.1}$$

En computer kan minimere det man kalder chikvadratet, der er et mål for hvor langt punkterne ligger fra modellen. Man kan bevise at det fit man får på denne måde er optimalt, og det er derfor meget ofte anvendt i den virkelige verden. Det er dog også



Figur 2.3: Almindelige funktioner plottet(tegnet) for forskellige værdier. Bemærk at det sidste plot er lavet som funktion af kvadratroden af x, kan du se hvordan man nu nemt kan aflæse parametrene a og b?

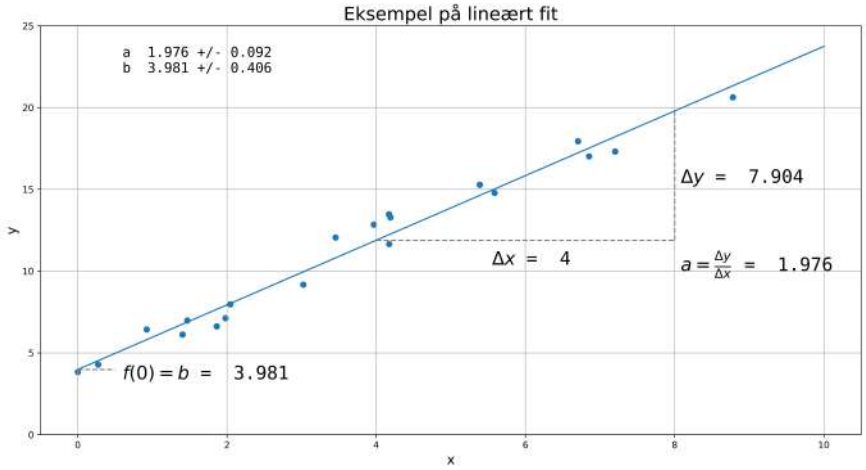
muligt at gøre dette 'i hånden', ved ganske enkelt at plotte punkterne på et stykke papir, og sætte en ret linje igennem punkterne[5] således at linjen følger punkterne generelt. Grundet små fejl og usikkerheder vil linjen ikke gå gennem alle punkterne, men den vil stadig beskrive den generelle tendens. Se figur 2.4 for et eksempel. Når man har denne linje, er det let at aflæse parametrene. Parameteren b fås ved at se hvor den skærer y-aksen. For at få parameteren a vælger vi to punkter på linjen, og noterer ændringen i x og y mellem dem. Parameteren a er nu givet ved forskellen i y delt med forskellen i x. Dette er også illustreret på figur 2.4.

Naturligvis vil den linje man sætter i hånden ikke være nøjagtigt den optimale linje som en computer kan finde, men man kan komme overraskende langt bare med øjemål - indtil computeren blev opfundet var der ikke andre muligheder.

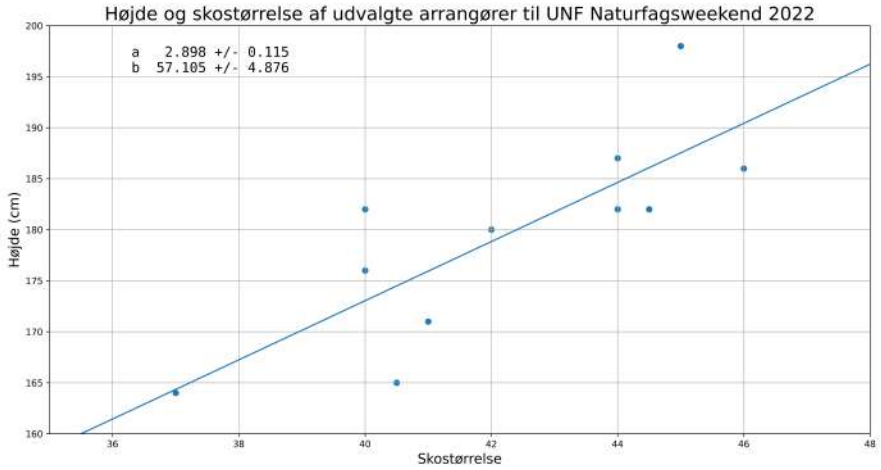
Eksempel

Vi vil nu kort vende tilbage til datasættet med skostørrelser og højder, for at vise et eksempel på et lineært fit til rigtige data. Se figur 2.5. Ved hjælp af en mængde data har vi konstrueret en lineær model for sammenhængen mellem skostørrelse og højde. Vi kan

[5]Med en lineal, gerne gennemsigtig.



Figur 2.4: Eksempel på lineært fit til datapunkter.



Figur 2.5: Arrangørers højde plottet sammen med deres skostørelse og fittet med en ret linje.

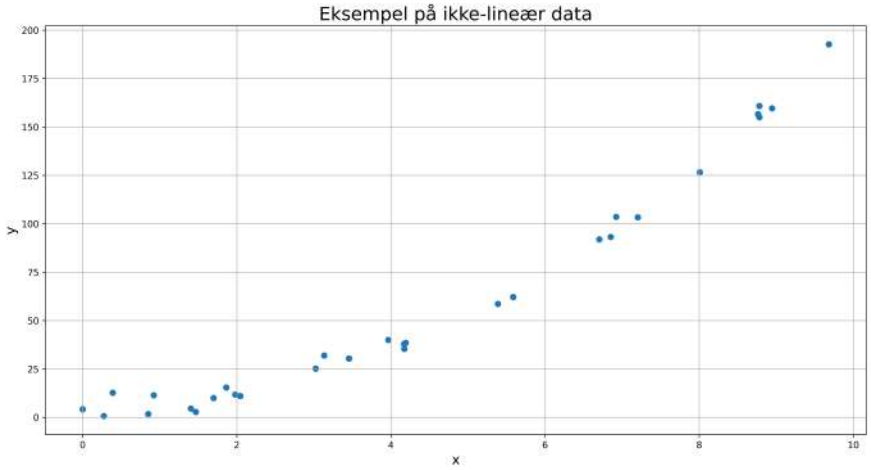
nu bruge denne model til at forudsige en persons højde ud fra deres skostørrelse, eller omvendt.

Transformation til lineær afhængighed

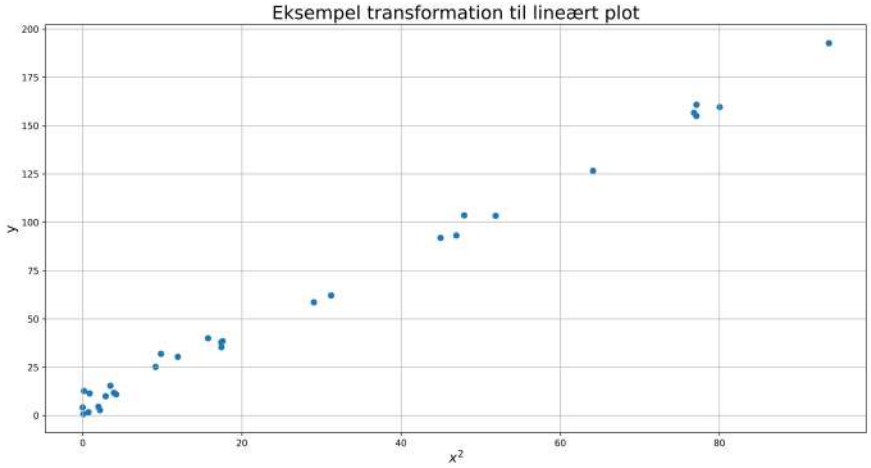
Er den sammenhæng man er interesseret i at undersøge ikke lineær, kan det være svært at lave et fit til data i hånden, da det er de færreste mennesker der kan tegne noget mere kompliceret end en ret linje. Her må vi gå mere snedigt til værks. Som altid er første skridt at plotte data. Har man gjort dette, kan man inspicere den og måske gennemskue hvilken sammenhæng der er mellem x- og y-akserne. Nu er hele ideen at man plotter sine data på sådan en måde, at der er en lineær sammenhæng mellem de nye akser. Nu kan man lave det lineære fit, og ekstrahere sammenhængen.

Eksempel

Ovenstående afsnit vil nok i første gennemlæsning virke meget abstrakt, og vi vil derfor gennemgå et konkret eksempel her. Betragt figur 2.6, her har vi et eksempel på en



Figur 2.6: Et eksempel på data hvor sammenhængen mellem x og y ikke er lineær.



Figur 2.7: Et eksempel på data hvor sammenhængen mellem x og y ikke er lineær, men hvor vi har transformeret den ene akse, således at plottet er lineært.

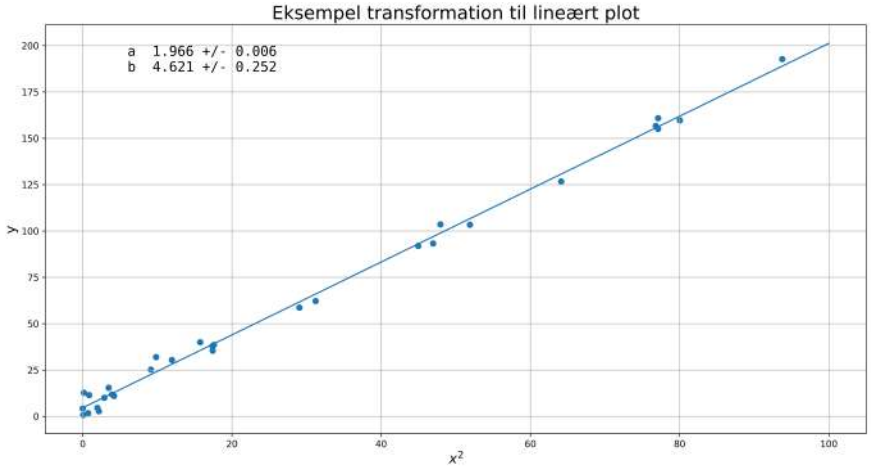
sammenhæng der tydeligtvis ikke er lineær (sammenlign med figur 2.3). Derimod kunne det godt se ud til at sammenhængen var noget i stil med:

$$y(x) = a \cdot x^2 + b \tag{2.2}$$

Nu kommer den vanskelige del, vi skal nemlig regne ud hvordan vi kan plotte y som funktion af x, således at sammenhængen på grafen bliver lineær. I dette tilfælde kan vi se at plotter vi y langs den ene akse og x^2 langs den anden vil grafen blive lineær. Dette er gjort på figur 2.7. Nu kan vi lave et lineært fit som før, ved blot at sætte en ret linje gennem, aflæse hældningen og aflæse skæringen. Se figur 2.8.

2.6 Opgaver

●● **Opgave 2.6.1: Hypoteser**
Vurder om de følgende hypoteser er gode og omformulér dem der er noget galt med så de bliver mere testbare. Hver opmærksom på at en hypotese ikke nødvendigvis er dårlig



Figur 2.8: Et eksempel på data hvor sammenhængen mellem x og y ikke er lineær, men hvor vi har transformeret den ene akse, således at plottet er lineært og har fittet en ret linje til den transformerede data.

fordi du ved at den ikke er sand. Det der gør en hypotese god er at den er specifik nok til at være nem at teste og f.eks. ikke er tvetydig.

1) Sten falder langsomt.

2) Des tungere en ting er desto hurtigere falder den.

3) Alle mennesker har en aura som kun marsmænd kan se.

4) Når man skubber til stole så falder de.

5) Stokroser overlever længere jo flere timer sol de får.

6) Der er ikke noget i rummet.

- **Opgave 2.6.2: Håndboldaften**
  Det er håndboldaften på skolen, hvad betyder det? At vi kan tage nogle skønne data selvfølgelig! Specifikt vil vi gerne vide hvordan antal mål en person scorer relaterer til hvor mange skud de tager. Vi har simpelthen skrevet ned hver gang der bliver skudt og om de så scorer. Vi har selvfølgelig anonymiseret vores data efter:
  Antal skud:[15, 26, 11, 12, 21, 23, 8]
  Antal mål:[9, 13, 6, 5, 9, 11, 3]

  1) Scorer man mere hvis man skyder mere?

  2) Hvor mange af ens skud scorer man ca.? (hint prøv at plotte det og lave et 'fit' med lineal)

- **Opgave 2.6.3: Mormors kageopskrift**
  Din mormor har givet dig hendes legendariske kageopskrift, men åh nej! I din hurtige løbetur hjem med opskriften, så har du tværet noget af opskriften ud. Der står hvordan alt fyldet laves og hvor lang tid den skal bages, men ingenting om ved hvilke temperatur. Du husker selvfølgelig præcis hvor fast den burde være på en skala fra 1 til 10. (Du kan også se på enhver kage og med det samme se hvor den falder på begge skalaer)

  1) Hvordan vil du finde den rigtige temperatur? (Du har en uendelig mængde mel og al den tid i verden du får brug for)

  I din glæde af at have fundet den rigtige temperatur så taber du opskriften ind i ovnen hvor den prompte brænder væk. Nu har du kun bagetid og temperatur. Men du kan ikke huske hvor meget af hver ingrediens der skulle i kagen for at skabe den smag

den har. (Du kan dog perfekt smage hvor meget af hver smag der er i kagen og husker den rigtige smag perfekt)

2) Hvordan finder du den rigtige mængde af hver ingrediens til fyldet? (Du har også en uendelig mængde af alle dine ingredienser Og du ved at hver en af dem kun påvirker en af smagsgrupperne)

●●● **Opgave 2.6.4: Perlearmbånd**

Du vil gerne lave perlearmbånd til hele din familie, men du vil helst have at der er lige mange perler på hvert armbånd og at der ikke er for langt mellem perlerne. For at se hvordan det ser ud med 20 perler på et armbånd laver du armbånd med forskellig radius og tæller hver gang hvor mange perler du får på 3 cm armbånd, du får følgende resultater:

Diameter af armbånd: [4, 6, 8, 12, 16, 22]
Perler på 3cm armbånd: [5, 3, 2, 2, 1, 1]

1) Plot datapunkterne, hvilken sammenhæng ligner det? Tjek om du har ret ved at lave et lineariseret plot.

●●● **Opgave 2.6.5: Vox pop, toastbrød, smør og tiptipoldefar**

I en vox pop, blev den gennemsnitlige befolkning spurgt om hvor høje deres tiptipoldefar var og på en skala fra 1 til 10 hvor godt de kunne lide toastbrød med smør. De fik så dette datasæt:

Tiptipoldefar højde (i cm): [172, 158, 167, 160, 183, 183, 179]
Hvor godt synes de om toastbrød med smør: [5, 9, 3, 7, 5, 9, 6]

1) Ren intuition, hvordan tror du de to datasæt afhænger af hinanden?

2) Hvor høje var folks tiptipoldefædre gennemsnitligt og hvor godt synes folk gennemsnitligt om toast med smør?

3) Kan du finde et fit der passer til datasættet? Hvad tror du det betyder?

Kapitel 3

Kemi

3.1 Introduktion

Velkommen til kemiforløbet på UNF's naturfagsweekend! I folkeskolen er de to fag fysik og kemi blandet sammen i ét fag Fysik/Kemi, men på f.eks. gymnasiet eller universitet er der kæmpe forskel på de to fag! Kemi handler om at forstå den måde universet atomare byggesten reagerer med hinanden, for at dét stof som hele verdens fysiske materiale består af. Mere lavpraktisk, så er det den del af naturfaget hvor man kan hælde to væsker sammen, hvorefter der sker en farveændring, en eksplosion, eller noget andet spændende!

I dette års undervisning/kompendium skal i lære de basale kemiske færdigheder der gør jer i stand til at regne på den kemi der skal i glasset nemlig *mængdeberegning*, og derefter to grene inden for kemien: *uorganisk kemi* og *termodynamik*.

Hvad er kemi for os?

Mette

Det, jeg finder interessant ved kemi, er, hvordan det kan forklare mange hverdags situationer. Man kan måske se, at der er sket noget som, at metal ruster og celler formerer sig. Bag begge fænomener er det kemiske reaktioner der finder sted. At finde ud af hvad der er sket, og hvordan den kan påvirkes er for mig det virkelig spændende. Stoffers kemiske egenskaber kan forklare hvorfor, stoffet reagerer, som de gør. For mig kan kemi give en forklaring af, hvordan ting hænger sammen og bruge kemi til at forbedre processer.

Knut

Kemi for mig er læren om verdens komponenter og hvordan, alt er bygget op. I kemi arbejder man på at kunne syntetisere nye stoffer og kunne detektere dem, dette gør man med et hav af analysemetoder. Kemi er derfor et naturvidenskabeligt fag, hvor al den viden, man har, kommer fra forsøg og eksperimenter. Det fantastiske ved kemi er, at det er med til gøre verden til et bedre sted og mange af de store problemers løsninger er baseret på viden fra kemi.

Ved hjælp af kemien kan man opskalere reaktioner og udvikle nye kemiske stoffer eller kunne producere stoffer, man tidligere fik fra naturen. Kemi hænger derfor meget sammen med industrien og har været en "katalysator" for industrialiseringen i de sidste 150 år. Blandingen af at finde ud af, hvordan verden er bygget op, og samtidig være en vigtig del af industrien er det, jeg synes, der gør kemi megafedt.

Lin

Kemi, i modsætning til så mange andre fag, er læren om reaktioner, og hvordan én ting kan ændre sig og blive til en helt anden ting. På den måde kan kemien beskrive alle mulige seje ændringer vi ser i hverdagen; som den ir der dannes på kobbertage,

eller hvordan fyrværkeriet på himlen nytårsaften fungerer, eller hvordan en tændstik og en tændstikæske lige pludselig kan lave ild! For mange år siden var en kemiker (en alkymist hed det den gang) mest beskæftiget med at fremstille uægte guld, mens nutidige kemikere med al den nye viden kan lave nye lægemidler eller designe store tanke når der skal produceres en masse håndsprit.

Det er dét jeg synes er fedt ved kemien: den kan beskrive alle de weird ting der sker omkring dig, og kan samtidig bruge sådanne beskrivelser til at hjælpe en masse mennesker - og det er bare super fedt.

3.2 Introduktion til atomer og molekyler

Atomet

Et atom er den mindste partikel der kan eksistere alene. Et atom består af en kerne, som er positiv ladet. Kernen består af 2 typer forskellige partikler: protoner og neutroner. Protonerene og neutroner er meget små og har en meget lille masse på $1.6710^{-27}kg$. Neutronen har en smule højre masse end protonen. Protoner er positive ladet og en ladning på $1.610^{-19}C$, som er den ladning man kalder for elementarladningen. Det vil sige at protoner har en meget høj ladningen i forhold til dens masse. Elektroner er negative ladet og er endnu mindre end protoner, og har en masse på kun $9.1110^{-31}kg$, men har samme størselorden af ladning som en proton.

Atom model

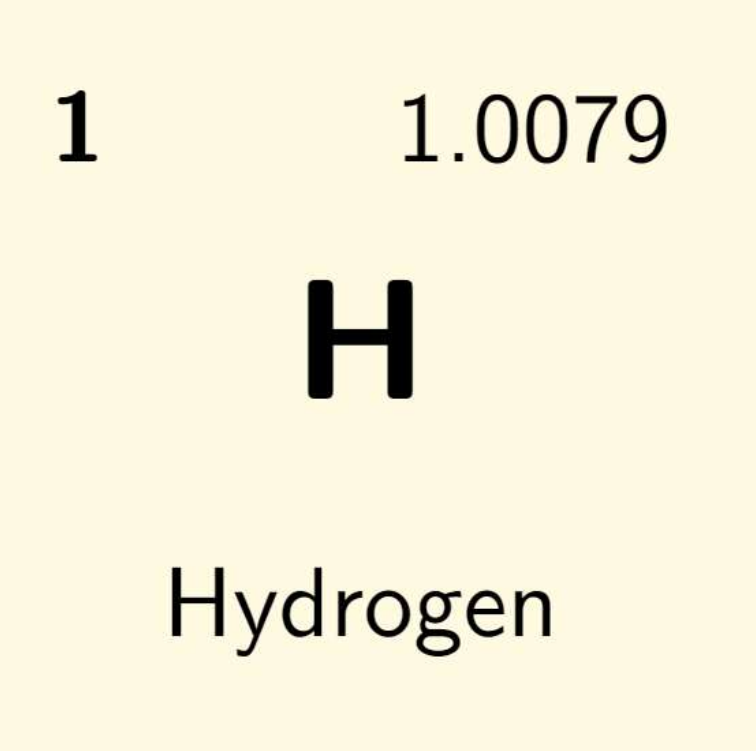
En god model af atomet er Bohrs atom model, som er en simplificering af virkeligheden. I virklhedden bevæger elektroner ikke baner, men det er en god approksimation til mange formål i kemien. Bohrs atom model består af en kerne hvor protoner og neutroner befinder sig i. Kernen er omgivet af "elektronbaner", hvor elektroner befinder sig i. Disse kaldes typisk for elektronskaller. I den inderste bane kan der være 2 elektroner, i den næste kan der være 8, derefter 18. Siden at kernen er positiv ladet og elektroner er negativt ladet, så vil elektronerne blive tiltrukket af kernen. Det kræver derfor energi at trække elektron væk fra kernen. Jo længere væk en elektron er fra kernen, jo høre potentielt enerig har den. Man kan drage parallel til tyngekraften, hvor det kræver energi at bevæge sig væk fra jordens kerne og potentielle energi vokser jo højre man kommer op. Det vil sige at elektroner bliver fyldt op i den inderste bane/skal først, og derefter i den næst-inderste også videre.

Grundstoffer og det periodiske system

Et grundstof er et atom med et bestemt antal protoner, hvor atomerne har lige mange elektroner og protoner når de ikke har en ladning. Hvis et atom mister eller får protoner bliver det til et nyt grundstof. Det sker i radioaktive processorer, men det er uden for kemiens verden. I kemi arbejder man kun med integrationen imellem elektroner, det vil sige mængden af grundstoffer er bevaret før og efter reaktion.

Det periodiske system er en oversigt over alle de kendte grundstoffer man kender til. Hvert grundstof har et nummer, nummeret repræsenter det antal af protoner som grundstoffet har. Omme bageste i dette kompendiet er der et printet periodisk system. Det første grundstof i det periodiske system er hydrogen, den har grundstof nummer 1 og består af et proton en elektron. Nogle hydrogen atomer har også en neutron. Antallet af neutroner der er i kernen kalder man for isotoper. For hydrogen findes der isotoper med 0, 1 eller 2 neutroner, som også kaldes for deuterium og trirtum. Dog er næsten alt hydrogen uden en neutron. På figur 3.1 kan du se et zoom af det periodisk system på hydrogen. Øverst til venstre er grundstof nummeret, til højre for det er den gennemsnitlige masse for hydrogen. Tallet er i units, hvor 1 unit er massen af en proton. Grunden til at massen af hydrogen er en smule højre end 1 er fordi der findes andre isotoper af hydrogen. Derudover har hvert grundstof en eller to bogstaver, som er en form for

forkorte for grundstoffet. Dette kalder man et grundstofsymbol, som f.eks. bruges til at skrive sammensætninger af molekyler på en kortere form.



Figur 3.1: Hydrogen i det periodiskesystem

Det periodiske system er bygget op efter hvor mange elektroner der i yderste skal og hvor mange skaller grundstoffet har. De grundstoffer der er øverste har deres yderste elektron i første skal, grundstoffer der har yderste elektroner i 2. skal, er også i anden række. Hver søjle i det periodiske system står for gruppe nummer,hvor dem der har tal øverste og går fra 1 til 8 er hovedgrupper, de andre er undergrupper.

Grundstoffer i hovedgruppe 1 har en elektron en i den yderste skal og grundstoffer i hovedgruppe 8 har 8 elektroner i den yderste skal.

Kemiske bindinger

Atomer eksisterer meget sjældent kun for sig selv, men indgår i stedet i forbindelser med andre atomer. Det sker i gennem en kemisk binding. Den følgende forklaring på kemiske bindinger er en forsimpling af virkeligheden. Bindinger dannes fordi atomer er mest stabile hvis deres yderste elektronbaner er fyldt op med elektroner. Dette kaldes også for oktetreglen, som siger at atomer skal have 8 elektroner i deres yderste skal for at være stabile. Dette er dog med undtagelse for H, He, Li og Be. Her er skal der være 2 elektroner i den yderste skal. Atomerne kan opnå dette ved at lave en kemisk binding. Der findes overordnet 2 forsklige typer af binding kovalente og ionske. Ioniske bindinger er hvor at elektroner bliver afgivet eller optaget af atomerne, fx kan Cl i vand optage en elektron fra Na og blive til Na^+ og Cl^-. Når Cl bliver til Cl^- er der 8 elektroner i den yderste skal, og oktetreglen er opfyldt, Na atomet af giver den yderste elektron og har 8 i den nu yderste skal og bliver til Na^+. Man kalder et atom der har flere eller færre elektroner end antallet af protoner for ioner.

Salte

Et salt er et neutralt ladet stof, som består af ioner. Det vil sige der er både er positive ioner og negative ioner, som udgør saltet. Et eksempel på et salt er køkkensalt NaCl, som består kationen (positiv ion) Na^+ og anionen (negativ ion) Cl^-.

Kovalente bindinger

En kovalnt binding sker når et atom indgår bindinger med et andet atom og der deles elektroner med et andet atom. Det fungere ved at begge atomer ser elektroner som deres eget. Dette gør fx at to hydrogen atomer kan danne en binding, og derved opfylde dubletregelen og blive stabil ved at danne H_2.

Molekyler

Et molekyle er sammensætningen af flere forskellige atomer, der er bundet sammen med kovalente bindinger. Man kan udtrykke molekyler ud fra en sumformel. Et eksempel på det er methan, som skrives som CH_4. Denne sumformel viser at molekylet består af et carbon atom og 4 hydrogen atomer, på grund af der er et nedsænket 4 tal efter. Når der kun er et af grundstoffet i molekylelet skriver man dog ikke et 1 tal, da det er implicit at der er én.

3.3 Reaktionsligninger og mængdeberegning

Introduktionen til reaktionsligninger

I kemien snakker man oftest om *reaktioner* mellem molekyler. En kemisk reaktion er, når molekyler reagerer med hinanden og danner nye molekyler. Et eksempel på en reaktion er forbrændingen af sukker, se reaktion 1

$$C_6H_{12}O_6 + 6\,O_2 \longrightarrow 6\,CO_2 + 6\,H_2O \quad \{1\}$$

Dette kaldes for et reaktionsskema eller en reaktionsligning. Det fortæller, at når der brændes ét $C_6H_{12}O_6$-molekyle (sukker), så bliver der brugt 6 O_2 (oxygen) molekyler , og der er blevet dannet $6\,CO_2$ og 6 H_2O.

Afstemning af reaktionsligninger

For at kunne regne på, hvad ændringen er af molekyler, er det vigtigt at reaktionsskemaet er *afstemt*. Dette betyder, er der er lige meget på begge sider, i forhold til masse, ladning og i forhold til mængden af de forskellige grundstoffer. Vi kigger på reaktionsligning (2)

$$N_2 + O_2 \longrightarrow NO_2 \quad \{2\}$$

Hvis reaktionen er afstemt, er der lige mange nitrogenatomer (N) og oxygenatomer (O) på venstre side af pilen, som på højre side af pilen. Lige nu er der 2 nitrogenatomer på venstre, og kun ét på højre, det fikser vi:

$$N_2 + O_2 \longrightarrow 2\,NO_2 \quad \{3\}$$

Men nu er der ikke nok oxygenatomer på venstre side, det fikser vi også:

$$N_2 + 2\,O_2 \longrightarrow 2\,NO_2 \quad \{4\}$$

Nu er der lige mange af N på højre og venstre side, og det samme med O. Så er reaktionsligningen afstemt!

Opgaver i afstemning af reaktionsligninger

- **Opgave 3.3.1:**
Afstem følgende reaktion (sørg for der er lige mange af hvert atom på hver side):

$$N_2 + H_2 \longrightarrow NH_3 \quad \{5\}$$

- **Opgave 3.3.2:**
Afstem følgende reaktion:

$$ZnS + O_2 \longrightarrow ZnO + SO_2 \quad \{6\}$$

Introduktionen til mængdeberegning

For at kunne regne på de reaktioner, man vil udføre, f.eks. hvor meget der skal tilføjes af et bestemt stof, er der brug for mængdeberegning. Mængdebregning er viden om, hvordan man "tæller" det antal molekyler, man arbejder med. Hvis man regner i per molekyler arbejder vil man komme til at arbejde med meget store tal, fx i en liter vand er der 34563375600000000000000000 eller eller $3,45 \cdot 10^{25}$ molekyler. For at gøre det meget nemmere at regne med, har man defineret enheden mol.

Stofmængde og mol

Stofmængde er, hvor mange molekyler man har af et bestemt stof. Til at beskrive antallet af molekyler bruger man enheden mol. Hvis man har et mol stof, f.eks. et mol af O_2 (ilt), har man $6,022 \cdot 10^{23}$ atomer. Man angiver altid stofmængder i enheden mol i kemi, på grund af, at det gør det meget nemmere at regne med.

Masse

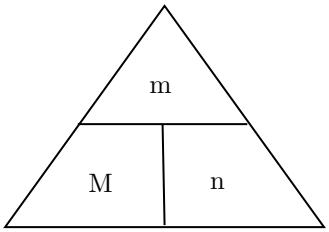
Masse er udtryk for, hvor meget materiale man har. I de fleste tilfælde og i hverdagen er massen og vægten det samme. Massen kommer fra de partikler, som atomet består af, det vil sige neutroner, protoner og elektroner. Hver af disse partikler har en bestemt masse. Masse måler man i kg, som står for kilogram. Her er gram enheden for massen, og kilo er et præfiks, der betyder 1000 gange. Det vil sige at der går 1000 gram på et kilogram. Det er samme princip inde for længde, som f.eks. med at på en kilometer går der 1000 meter.

Molarmasse

For at kunne udregne stofmængden af et bestemt stof har man brug for at kende molarmassen. Molarmassen fortæller, hvor mange gram stof der skal til, før man har et mol af det. F.eks. vejer 1 mol jern (Fe) 55,845 g. Man finder molarmassen ved at bruge det periodiske system. Da det, der giver massen kommer fra de partikler, som atomet består af, kan man finde massen af et atom ved at lægge massen af alle protonerne, neutronerne og elektronerne sammen.

Eksempel på beregning af carbons molarmasse

Carbon er nr. 6 i det periodiske system, det vil sige at det indholder 6 protoner og 6 elektroner. Afhængigt af isotopen har carbon 6, 7 eller 8 neutroner, langt de fleste carbonatomer har 6 neutroner. Det vil sige, at et carbonatom generelt har 6 protoner, 6 neutroner og 6 elektroner. En proton har en masse på $1{,}66 \cdot 10^{-27}$ kg. Massen af en mol protoner er på præsis 1 g. Dette kan man se ved at gange massen af et proton med Avogadros tal (antallet af molekyler i et mol stof), som kan ses i ligning (3.1)



$$M_{proton} = m_{proton} \cdot N_A = 1{,}66 \cdot 10^{-27}\,\mathrm{kg} \cdot 6{,}022 \cdot 10^{23}\,\mathrm{mol}^{-1} = 1\,\frac{\mathrm{g}}{\mathrm{mol}} \tag{3.1}$$

Protoer og neutroner masse næsten er ens og elektroner masse er meget lille i forhold til er masse af et mol carbon atomer 12g. Molarmassen af et bestem af atom er massen af 1 mol af atomet. Det vil sige at molarmassen af carbon er 12 $\frac{\mathrm{g}}{\mathrm{mol}}$. Molarmassen af alle grundstoffer står i det periodiske tabel. Man kan finde molarmassen af et molekyler ved at ligge molarmssen hvor hvert atom i molekyllet sammen.

Eksempel på beregning af vands molarmasse
Vand(H_2O) består af 2 Hydrogen atomer og et oxygen atom. Molarmassen af Hydrogen er 1 $\frac{\mathrm{g}}{\mathrm{mol}}$ og molarmassen af oxygen er 16 $\frac{\mathrm{g}}{\mathrm{mol}}$.Der tages summen af 2 gange molarmmsen af hydrogen og en gang af molarmassen af oxygen, som ses i ligning (3.2)

$$M_{\mathrm{H_2O}} = 2 \cdot M_H + 1 \cdot M_O = 2 \cdot 1\,\frac{\mathrm{g}}{\mathrm{mol}} + 1 \cdot 16\,\frac{\mathrm{g}}{\mathrm{mol}} = 18\,\frac{\mathrm{g}}{\mathrm{mol}} \tag{3.2}$$

Sammenhæng mellem stofmængde, molarmasse og masse

I afsnittet før er der blevet introduceret tre grundværdier inde for mængdeberegning:

- Stofmængde: hvor mange molekyler der er, symbol n
- Masse: vægten af stoffet, symbol m
- Molarmassen: massen af 1 mol af et vilkårligt stof, symbol M

Når to af disse værdier kendes, kan man finde den sidste. Sammenhængen mellem dem ses på den følgende skitse.

Ligning 3.3 viser sammenhængen mellem stofmængde, molarmasse og masse

$$n = \frac{m}{M} = \text{stofmængde} = \frac{\text{masse}}{\text{molarmasse}} \tag{3.3}$$

For at kunne finde én af de tre, skal man kende de to andre. Hvis man f.eks. kender vægten en en klump jern, og ved at jerns molarmasse er 55,845 g/mol kan man finde stofmængden. Der vises et eksempel på, hvordan man finder molarmassen ud fra, at man kender massen og stofmængden.

Eksempel på bestemmelse af molarmasse for en ukendt blanding gas
Man vil undersøge en gas for, om det er methan (CH_4) eller ethan (C_2H_6), hvor man har kunne køle gassen ned og målt massen af den til at være 57 g, og ved hjælp af en trykmåling er det fundet, at der er 3,4 mol gas. Der findes først molarmassen af methan og ethan.
Methan består af ét carbonatom og 4 hydrogenatomer og har en molarmasse på

$$M_{\mathrm{CH_4}} = 12\,\mathrm{g/mol} + 4 \cdot 1\,\mathrm{g/mol} = 16\,\mathrm{g/mol}$$

og ethan består af 2 carbonatomer og 6 hydrogenatomer og en molarmasse på:

$$M_{C_2H_6} = 12\,\text{g/mol} \cdot 2 + 6 \cdot 1\,\text{g/mol} = 30\,\text{g/mol}$$

Nu findes molarmassen af den undersøgte gas, og derefter bliver den sammenlignet med molarmassen af ethan og methan. Der bruges ligning (3.3) til at isolere for molarmassen:

$$M = \frac{n}{n} \cdot M = \frac{m}{M} \cdot M \cdot \frac{1}{n} = \frac{m}{n} \tag{3.4}$$

De kendte værdier indsættes i dette:

$$M_{gas} = \frac{m}{n} = \frac{57\,\text{g}}{3{,}4\,\text{mol}} = 16{,}7\,\frac{\text{g}}{\text{mol}} \tag{3.5}$$

Det vil sige, at den ukendte gas består primært af methan, da 16,7 er tættere på 16 end 30.

Opgaver i mængdeberegning

- **Opgave 3.3.3:**
  Bestem molarmassen af H_2O (vand) med hjælp fra et periodisk system.

- **Opgave 3.3.4:**
  Bestem molarmassen af $CuFeS_2$ (kobberpyrit) med hjælp fra et periodisk system.

- **Opgave 3.3.5:**
  Et stykke jern vejer 5 g (dvs. $m = 5g$), hvad er stofmængden af jern? (n =?). Brug molarmassen af jern $M = 55,8\frac{g}{mol}$.

- **Opgave 3.3.6:**
  Der haves 2 g CaO, hvad er stofmængden af CaO?

- **Opgave 3.3.7:**
  Der haves 7 mol Hg, hvad vejer denne væske?

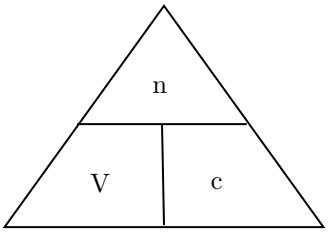
- **Opgave 3.3.8:**
  Lin har fundet en gråt metal på jorden uden for Sukkertoppen gymnasium. Hun har vejet det og konstaterer at klumpen vejer 10,0 g, og med sine seje kemividen ved hun at stofmængden er 0,178 mol. Hvilket metal er der tale om?

Koncentrationer

Når stofmængder er begrænset til et rumfang, kan det betegnes som koncentration. Hvis man f.eks. har en beholder med 1 liter vand og 1 mol af et stof opløst i vandet, er der således en koncentration på 1 mol pr. liter, og den samme koncentration kan opnås hvis man har 2 mol i 2 liter vand. Der er således lige mange molekyler pr. plads. Koncentration kan regnes med følgende formel

$$c = \frac{n}{V} \tag{3.6}$$

Hvor n er stofmængden, V er rumfanget og c er koncentrationen som er målt i enheden molær (M), som betyder mol pr. liter. Sammenhængen mellem n, V og c kan ses på følgende skitse



Figur 3.2: Regnetrekant over koncentrationer

Fordelen ved koncentrationer er, at hvis koncentrationen er kendt, vil man kunne bestemme stofmængden ud fra et afmålt rumfang. *Eksempel på beregning af koncentrationen af salt i en saltopløsning Hvis 2 gram NaCl (bordsalt) opløses i 2 L vand, hvad vil koncentrationen af NaCl så være?
Først beregnes stofmængden. Molarmassen af NaCl er 58,5 $\frac{\text{g}}{\text{mol}}$

$$n = \frac{m}{M} = \frac{2\,\text{g}}{58{,}5\,\frac{\text{g}}{\text{mol}}} = 0{,}0342\,\text{mol}$$

Koncentrationen beregnes ud fra formlen $c = \frac{n}{V}$

$$c = \frac{0{,}0342\,\text{mol}}{2\,\text{L}} = 0{,}0171\,\text{M}$$

Koncentrationen af NaCl i saltopløsningen er 0,0171 M (som betyder mol per. liter).

Opgaver i koncentrationer

- **Opgave 3.3.9:**
  10 mol NaCl (bordsalt) er blevet hældt i et POKAL-Ikea glas på 0,35 L. Hvad er koncentrationen af salt i vandet?

●●● **Opgave 3.3.10:**
Ifølge pålidelige kilder kan et menneske drikke saltvand (NaCl i vand) med 180 $\frac{mg}{L}$. Kan man drikke glasset med saltvand fra forrige opgave?

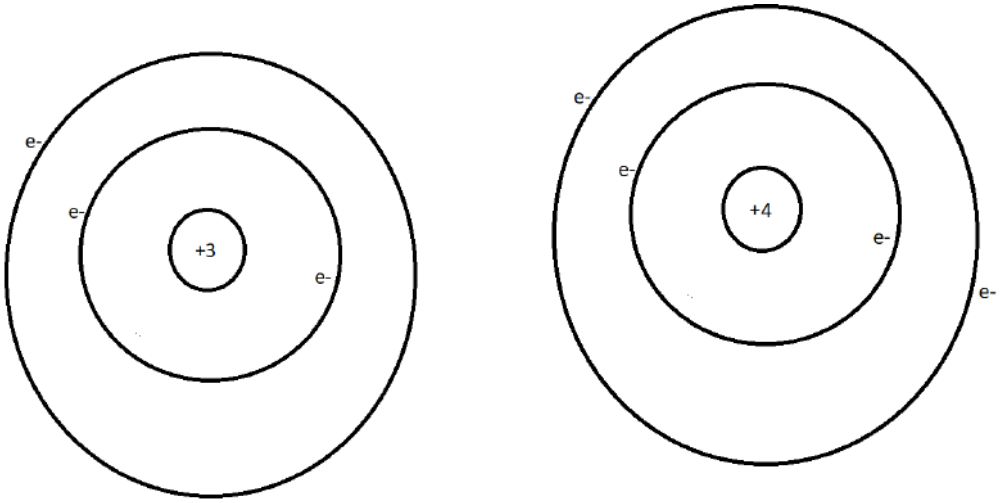
●●● **Opgave 3.3.11:**
Der er vejet 5g $KMnO_4$ af i en målekolbe, og fyldt demineraliseret vand op til 100 mL. Hvad er koncentrationen af $KMnO_4$ i opløsningen?

3.4 Uorganisk kemi

Uorganisk kemi er den del af kemi der beskæftiger sig med alt der ikke er organiske kemi, det vil sige alle molekyler der ikke består af carbon kæder. Det vil sige at i uorganiskekemi dækker over langt de fleste kemisker stoffer. I organiskekemi kommer mange af startstoffer fra olie, hvor at uorganisk kemi kommer kemikalier fra mange forsklige kilder alt fra mine, havet og mere. Mennesket krop indholder også en del uorganisk kemi, blandt andet hæmoglobin som er et enzym der gør at vores bold bliver iltet,og salt i sveden. Uorganisk kemi bliver også meget anvendt indefor både heterogene katalysatorer (2 faser) og homogene katalysatorer (1 fase). I dette afsnit vil der blive set næremere på uorganisk forbindelser i opløsninger.

Det periodiske system

Grundstoffer der er i samme hovedgruppe minder om hinanden, fx så reagerer alle grundstoffer i første hovedgruppe, bortset for hydrogen, meget kraftigt når de bliver smidt i



(a) Bohrs atommodel for Litium, hvor kernen er simplificeret til kun summen af protonerne.

(b) Bohrs atommodel for Beryllium, hvor kernen er simplificeret til kun summen af protonerne.

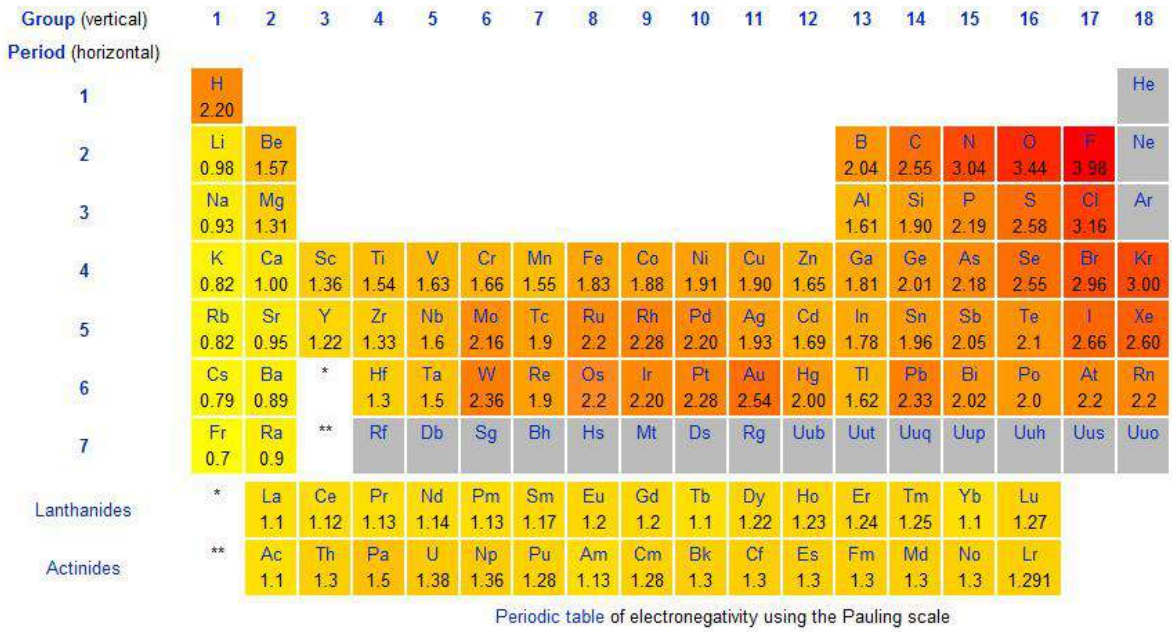
Figur 3.3: Bohrs atommodel for Beryllium og litium, for at visse

vand. Man kalder grundstofferene i første hovedgruppe, bortset hydrogen, for alkalimetaller. Alkalimetaller kendes ved at de er metalliske, bløde og findes enten som metal eller som en ion med +1 ladning. Grundstofferene i anden hovedgruppe kaldes for jordalkalimetaller og har mange ligheder med alkalimetaller, dog danner de primært +2 ioner og er mindre reaktive.
Til højre for anden hovedgruppe har man overgangsmetallerne.

Elektronegativitet

Elektronegativitet er et begreb der beskriver hvor meget kernen fra et givet grundstof trækker i elektroner fra et andet atom. Den følgende forklaring er igen en forsimpling af virkeligheden, men er en god start til at forstå kemiske egenskaber ved de forsklige grundstoffer. Man kunne tro at elektronegativiteten er den samme for alle grundstoffer, siden alle grundstoffer er neutralt ladet. Det der giver effekten, er at elektroner der er skallen før skærmer for kernen på de yderste elektroner. Det som den yderste elektron mærker af kernen kalder man den **effektive kerneladning**. Et eksempel på dette er når man kigger på Lititum og Berylyium, som er grundstof nr. 3 og 4, som man ses 3.3. I litium er der to elektroner i den inderste skal som skærmer for for den yderste elektron, i beryllium er der også to elektroner i skallen, som før men ladningen fra kernen er én højere. Det vil sige elektroner i den yderste skal bliver mere påvirket af kernen i beryllium end i litium. Det vil sige at beryllium er mere elektronegativ end litium. Dette gør også at beryllium er mindre i størrelse end litium.

På figur 3.4 kan man se elektronegativiteten af alle kendte grundstoffer. Her ses det at når man bevæger sig til højre i det peridoskesystem bliver elektronegativiteten større. Dette er fordi at der bliver fyldt op i samme elektronskal, samtidig med at kernens ladningen bliver større. Det vil sige at de yderste elektroner bliver mere tiltrukket af kernen og elektronegativiteten stiger. Desto længere man bevæger sig ned i det periodiske system, det vil sige at man kigger på grundstoffer med flere skaller, falder elektronegativiteten. Dette kan simpelt forklares med at de yderste elektroner er længere væk fra kernen og der er flere lag af elektroner til at skærme for den yderste skal.



Figur 3.4: Diagram over elektronegativteten af de forsklige grundstoffer [6]

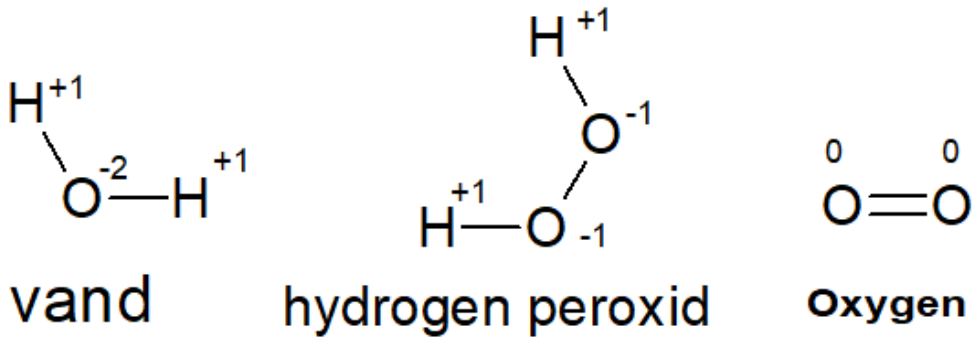
Kemisk betydning for grundstoffet

En høj effektiv kerne ladning gør at grundstoffets radius bliver mindre, da kernen har en større effekt på de yderste elektroner, og at den derfor trækker de yderste elektroner tættere på. For grundstoffer i hovedgruppe 7 (halogene), gælder det at grundstofferne har 7 elektroner i den yderste skal, og mangler én for at kunne opfylde oktet reglen. Det vil sige at kernen skal kunne tiltrække en elektron fra et andet atom, hvilket så betyder at jo højere effektiv kerneladning: jo mere reaktiv er grundstoffet i hovedgruppe 7. Fluor er derfor det mest reaktive grundstof i gruppe 7. Astat er det mindste reaktive, da den har en noget lavere effektiv kerneladning og dermed sværere ved at danne At^--ionen.
For grundstoffer i hovedgruppe 1, er det lige omvendt, fordi her skal grundstoffer smide en elektron for at blive mere stabile. Desto mindre kernens effekt er på den yderste elektron jo nemmere er det at fjeren den. Jo lavere den effektive kerneladning er, jo mere reaktiv er grundstofferne i 1. hovedgruppe.

Oxidationstal

Oxidationstal er en betegnelse for hvor elektroner formelt er placeret i en forbindelse. Man kan se det som en form for "lokal ladning" på atomerene i en kemiske forbindelse og er et udtryk for hvilken af atomkerne elektroner i en kemiske binding er tættes på. Hvis oxidation tallet er positiv betyder det atomet har færre elektroner end, som rent grundstof.
Hvis oxidation tallet er negativ betyder det at atomet har flere elektroner end som rent grundstof. Oxidationtallet for atomer i molekyler kan man finde ved at bruge elektronegativitet, da det er et udtryk på hvor meget det atomet trækker i andre atomers elektroner.
Eksempel
Hvis man ser på et molekyle som H_2O(vand), det består af 3 atomer, hvor der er to kemiske bindinger mellem hvert hydrogen (brint) og oxygen (ilt). Man ser på to grundstoffer i bindigen her har hydrogen en en elektronegativitet på 2,2 hvor at oxygen har en på 3,44. Det vil sige at elektronene i bindingen ligger tættest på oxygen og dermed bliver oxidationtallet for hydrogen +1, da den elektron den har i binding bliver set som at være væk fra hydrogenatomet, da oxygen har 2 af disse bindinger har den et oxidationstal af

-2, da den har en ekstra elektron for hvert af de to bindinger.
I et molekyle skal summen af oxidationstale for alle atomer være lig med ladningen af molekylet. I eksemplet med vand er der 2 hydrogen med et oxidationtal på +1 og et oxygen med -2, det vil sige summen af oxidation tal er lig med 0 og det passer med ladningen af vand er 0.
I molekyler som kun består af et grundstof er oxidationtal altid nul, fx H_2 der er oxidation tallet af begge hydrogen atomer 0, da hvert hydrogen "trækket" lige hårdt i elektroner i bindingen.



Figur 3.5: Strukturer med oxidation tal for atomerne i tre forsklige forbindelser med oxygen. De tre forbindelser fra venstre til højre er vand, hydrogenperoxid og ilt

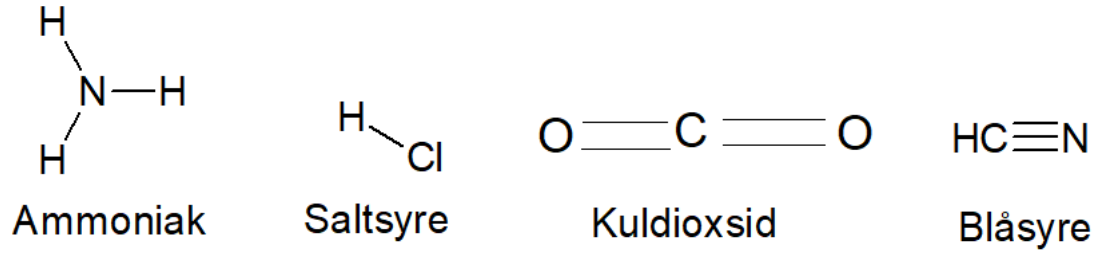
Eksempel
Hydrogen peroxid(H_2O_2), kan bruges som et oxidationsmiddel, det består af 2 hydrogen atomer og 2 oxygen atomer, hvor hvert oxygen er bundet til et anden oxygen og til et hydrogen, som vist i figur 3.5. I binding mellem hydrogen og oxygen ligger elektronparret tættes på oxygen og hydrogen får +1 i oxidation tal og oxygen får fra den -1 oxidationtal, da det er to oxygen atomer der er bundet sammen trækker de lige hårdt i elektroner og det bidrager med nul til oxidation tallet af hvert oxygen atom og oxidationtallet af oxygen i hydrogenperoxid bliver -1.

- **Opgave 3.4.1:**
  Hvad er oxidationstallet af nitrogen og hydrogen i ammoniak(CH_3), strukturen kan ses på figur 3.6?

- **Opgave 3.4.2:**
  Hvad er oxidationstallet af klor(Cl) og hydrogen i saltsyregas(HCl), strukturen kan ses på figur 3.6?

- **Opgave 3.4.3:**
  Hvad er oxidationstallet af kulstof(C) og ilt(O) i kulstofdioxsid(CO_2), strukturen kan ses på figur 3.6?

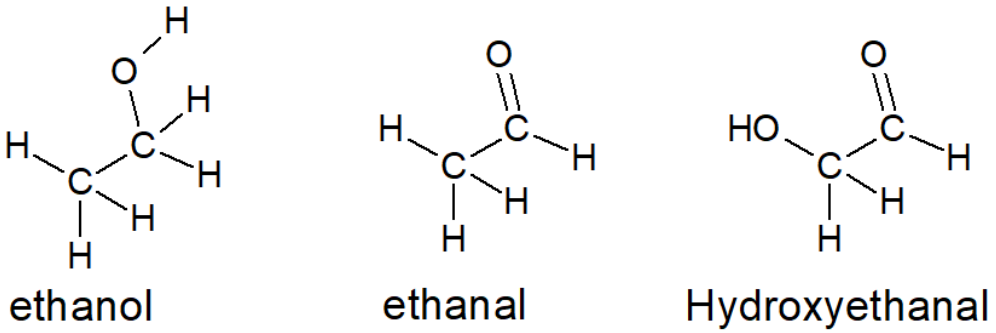
- **Opgave 3.4.4:**
  Hvad er oxidationstallet af kulstof(C), kvælstof(N) og brint(H) i blåsyre(HCN), strukturen kan ses på figur 3.6?



Figur 3.6: Strukturformel af ammoniak, saltsyre, kuldioxid og blåsyre, der skal anvendes til opgaver.

Samme grundstoffer med forkslige oxidationstal i et molekyle.

Oxidationstallet af et atom i et molekyle bestemmes som vist tidligere ud fra de bindinger der er til atomet, det vil sige, at der kan være flere af det samme grundstof i et molekyle med forskellige oxidation tal.
Eksempel
Det stof i øl der gør man bliver påvirket er ethanol(CH_3CH_2OH, i dagligt tale kaldt for alkohol), strukturen kan er vist i figur 3.7. Ethanol har 2 kulstof atomer, 6 hydrogen og et oxygen atom. Den består af 5 C-H bindinger, 1 C-C binding, 1 C-O og en O-H binding. Fra figur 3.4 kan man aflæsse at oxygen er mest elektronegativ, derefter carbon, og hydrogen er mindst elektronegativ. Det vil sige at alle hydrogen atomer har oxidation tal +1 i ethanol, carbon atomet der er bundet til 3 hydrogen og et carbon har oxidation tallet -3, da den får en elektron for hvert af hydrogen atomerene og fra C-C bindingen trækker kulstof atomer lige hård og har ingen indflydelse på oxidation tallet, det andet kulstof atom har oxidation tal -1 da den får 2 elektroner fra de 2 H-C bindinger, 0 fra C-C og mister en i C-O bindingen. Oxygen har oxidation tal -2 em fra O-H og en fra O-C bindingen.



Figur 3.7: Sturkturformel for ehtanol, ethanal og hydroxyethanal.

- **Opgave 3.4.5:**
  Hvad er oxidationstallet for de to kulstof atomer i ethanal(CH_3CHO)? Strukturen kan ses på figur 3.7.

- **Opgave 3.4.6:**
  Hvad er oxidation tallet af de to kulstof atomer i hydroxyethanal(CH_2OHCHO)? Strukturen kan ses på figur 3.7.

Oxidationstal i salte

Salte består af en positiv ion (kation) og en negativ ion (anion), man kan finde oxidationstallet af ioner af grundstoffer ved at det simpelt er lig med ladning. **Eksempel** Den

simple ion til natrium er Na^+, da summen af oxidation tal for en forbindelse skal være lig med ladning og Na^+, kun har et atom er oxidation tallet af Na være 1. En generel regel er at hvis noget er bundet til oxygen (bort set fra fluor og oxygen) har det oxygen oxidation tal -2 og man ser kun O-O forbindsler meget relative forbindelser og ikke i metaller. I salte opskrives forbindelser tit som Fe_2O_3, det kan ligne det er et molekyle med 2 jern og 3 oxygen men i virklige er det et meget store saltgitter hvor forholdet imellem jern og oxygen er 2 til 3. Oxidation tallet for jern i den forbindelse er +3, da oxidation tallet for oxygen er -2 og der er 3 oxygen atomer og saltet er neutralt, skal summen af de 2 jerns oxidation tal være lig med $-1(3 \cdot -2) = 6$, det vil sige oxidationtallet er +3 for jern i den forbindelse.

- **Opgave 3.4.7:**
  Hvad er oxidationstallet for Mg^{2+}?

- **Opgave 3.4.8:**
  Hvad er oxidationstallet for CaO?

- **Opgave 3.4.9:**
  Hvad er oxidationstallet for K^+?

- **Opgave 3.4.10:**
  Hvad er oxidationstallet for MnO_2?

- **Opgave 3.4.11:**
  Hvad er oxidationstallet for $Mn_2{}^{2+}$?

- **Opgave 3.4.12:**
  Hvad er oxidationstallet for $MnO_4{}^-$?

- **Opgave 3.4.13:**
  Hvad er oxidationstallet for KMn_{04}?

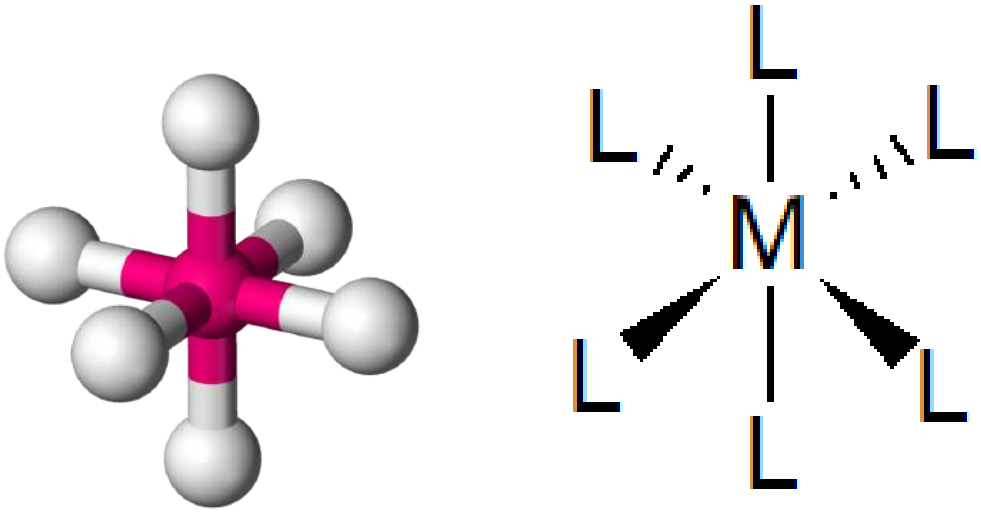
- **Opgave 3.4.14:**
  Hvad er oxidationstallet for $Cr_2O_7{}^-$?

3.5 Komplekser

De to bindningstyper, som er blevet introduceret indtil videre, har været kovalente bindinger og ionbindinger. Der findes en tredje type binding kaldet kompleksbinding. Det kan både være et negativt eller neutralt ladet molekyle og et positivt metalion. Den positive metalion kan have flere af disse kompleksebindinger, dette kalder man får et kompleks. De molekyler/ioner der er bundet til metalion kalder man for **ligander**. Et kompleks har ofte 6 eller 4 ligander, altså at metalionen er bundet til 6 eller 4 molekyler ellerioner. For at et molekyle kan være en ligand skal det have elektroner den kan afgive til metalionen. Komplekserene er bygget op ved at der er en positiv metalion centreret i midten, som liganderne er bundet til. Når man snakker om hvordan liganderne er bundet til metalionen og hvordan de sidder forhold til de andre ligander, snakker man om hvordan liganderne er *koordineret*. Hvordan ligander koordinerer til metalionen afhænger af tre ting: metallet, oxidationstallet og liganden selv.

Strukturer af komplekser

For komplekser med 6 ligander kaldes strukturen for oktaedrisk. Det karakteristiske ved denne struktur (geometri) er at de 6 ligander er delt ud på tre akser og i hver deres ende af akserne. Det vil sige at i et 3d-koordinatsystem så ligger to af liganderne på x-aksen, 2 på y-aksen og 2 på z-aksen. For hvert par er de to ligander placeret på hver deres side af metalionen. På figur 3.9, er der vist hvordan man tegner strukturen generelt for et oktaedrisk kompleks.

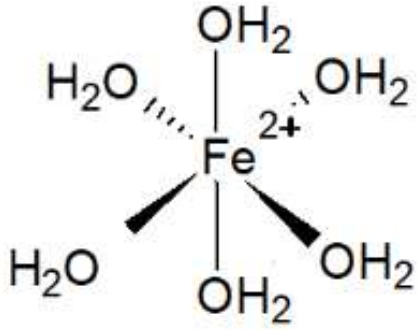


(a) Generel struktur af et oktaedrisk kompleks, tegnet 3 dimensionelt. Hvor de hvide kugler er ligander og den lyserøde kugle er metalionen.

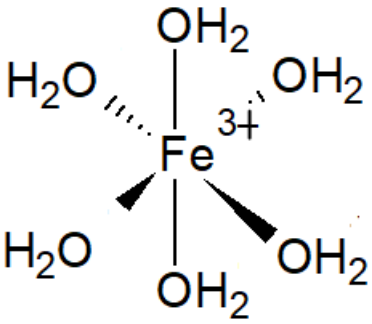
(b) Generel 2 dimensionel struktur af et oktaedrisk kompleks. Hvor L er ligand og M står for metalionen. Stregen indikerer binding imellem metalion og ligand.

Figur 3.8: 3 dimensionel struktur og 2 dimensionel struktur af et 6 koordineret kompleks

Måden man skal læse figur 3.10b på er at hver trekant/streg er en kompleks binding, de fyldte trekanter betyder, at bindingen vender mod læseren, de stiplede betyder at bindingen vender ind i papiret, og en streg betyder at bindingen er langs med papiret. Sammenligningen mellem figur 3.10a og 3.10b kan hjælpe på at bedre visualisere tegningen. Oxidationstallet af metalionen i et kompleks har stor betydning. I figur 3.10a og figur 3.10b, er der to forsklige komplekser som har store ligheder, bort set fra at oxidationstallet af jern er forskelligt, denne forskel gør at farven af de to komplekser er vidt forskelige og jern(III) komplekset opløsningen bliver væsentligt mere sur.



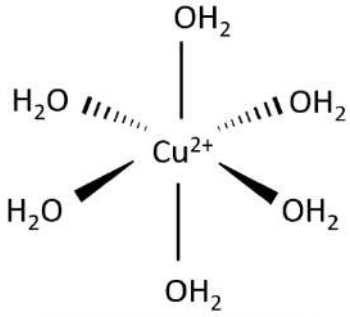
(a) Struktur af hexaaquajern(II) ion.



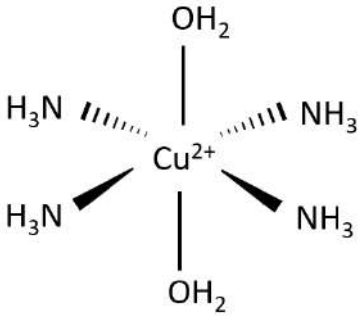
(b) Struktur af hexaaquajern(III) ion.

Figur 3.9: Struktur af 2 jernkomplekser med ens ligander,men forsklige oxidationstal.

Et anden måske komplekset kan få en anden farve på at ved at der sidder forskellige ligander. Fx er $[Cu(OH_2)_6]^{2+}$ lyseblåt, hvorimod $[Cu(NH_3)_3(OH_2)_2]^{2+}$ er mørkeblåt, se figur 3.10. Der er samme ladning, det er samme metalion, men de forskellige ligander gør at det meget mørkeblå kompleks optager mere af det gule lys.



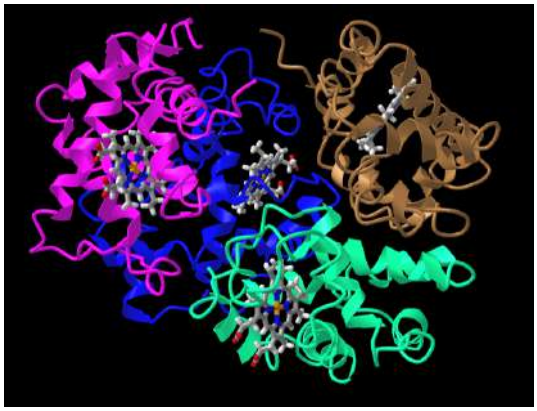
(a) Struktur og farve af hexaaquakobber(II) ion.



(b) Struktur og farve af tetraamindiaquakobber(II) ion.

Figur 3.10: Struktur af 2 kobberkomplekser med forskellige ligander, men samme oxidationstal. Farvene af komplekserne er dog stadig forskellige.

De komplekser vi har vist før er meget simple. I kemi bliver komplekserne tit mere komplicerede, hvor ligander fx kan finde på at lave flere bindinger til metalionen. I biologi er det noget værre rod, med kæmpe-molekyler (proteiner) der kan være svære at holde styr på. I figur 3.11 se et billede af hæmoglobin - det protein i kroppen der binder ilt og transpoterer det rundt i blodet. Det er de såkaldte hæmgrupper der binder ilten, og hæmgrupperne er komplekser!



Figur 3.11: Struktur af proteinet hæmoglobin. Hæmgrupperne (komplekserne) er tegnet med tydelige grå streger. Der er her ilten bliver bundet.

3.6 Termodynamik

Termodynamik er en del af kemien der grænser op i mod fysikken. Termodynamik er læren om relationer mellem temperatur i form af varme og energi i form af arbejde.En reaktion foregår i et system, her kan der kan der under reaktionen ændres på tryk og volumen, hvilket gør der bliver udført noget arbejde. Arbejdet kan bevirke at der dannes ekstra varme. Der skal energi til førend en reaktion forløber, der kaldes for Gibbs energi. Gibbs energien afhænger af entalpi og entropi.

Entalpi

Etalpi beskriver energie i for en reaktion. Der vil ofte enten absorberes (optages) energi eller frigives energi under en kemisk reaktion. Hvis der ses på et system med en beholder, så skal reaktionen enten tage noget af varmen udenfor systemet eller frigive ekstra varme til systemet. Entalpi beskrives med symbolet H og ved en ændring ΔH det vil sige forskellen i enhalpi mellem start og slut for reaktionen. Entalpi har enheden kJ, hvor k er et præfiks for 1000 enheder. Entalpien afhænger af den indre energi det vil sige selve systemets energi U samt tryk og volumen af reaktionsblandingen $\Delta(p \cdot V)$ (se ligning 3.7).

$$\Delta H = \Delta U + \Delta(p \cdot V) \tag{3.7}$$

Når en reaktion forløber vil der ofte blive ændret volumen og tryk, idet produktet/produkterne fylder mere eller mindre end reaktanterne gjorde. Energien vil derfor blive påvirket, idet der mulighed for sammenstød og kræft ændret. Om en reaktion frigiver eller absorbere energi, påvirker om reaktionen danner varme eller tager varme fra omgivelserne. Reaktionen kaldes eksoterisk, hvis den afgiver energi til omgivelser. Her er ΔH mindre end 0, idet energien går ud af systemet.

$$\Delta H < 0$$

Hvis reaktionen derimod optager energi kaldes den for endetermisk. Her er ΔH større end 0, idet der tilføres energi til systemet.

$$\Delta H > 0$$

Regneeksempel med enthalpi

Nu hvor alt det teoretiske er gennemgået så gennemgås et eksempel på hvordan man bestemme enthalpien for en reaktion. For at beregne enthalpien anvendes følgende formel.

$$\Delta H = \Sigma(H_{produkter}) - \Sigma(H_{reaktanter}) \tag{3.8}$$

$$C_3H_8(g) + 5O_2(g) \rightarrow 3CO_2(g) + 4H_2O(g)$$

Tegnet Σ er det græske symbol sigma og beskriver en sum. Det vil sige det er alle produkters enthalpi lagt sammen minium alle reaktanterens enthalpi lagt sammen. Der ses på en forbrænding af propan (C_3H_8) og oxygen (O_2) til kuldioxid (CO_2) og vand (H_2O):

Her er det vigtig at man holder øje med, hvilken tilstandsform hvert stof har, idet deres enthalpi afhænger af tilstandstformen. Som et eksempel med vand er som gas $\Delta H(H_2O(g)) = -248,8kJ$ og som flydende $\Delta H(H_2O(l)) = -285,83kJ$.
Som det første findes enthalpiværdierne for alle stofferne i reaktionen. Dette kan ses i tabellen nedenfor. Hermed er der alle værdier for at kunne beregne ændringen i ent-

Molekyle	Δ H
C_3H_8 (g)	$-103,85kJ$
O_2 (g)	$0kJ$
CO_2 (g)	$-393,51kJ$
H_2O (g)	$-248,80kJ$

Tabel 3.1: Enthalpi værdier for forbrænding af propan

halpien over reaktionen. Her skal ´man være opmærksom på reaktionens støkiometrisk forhold, det vil sige hvor mange molekyler der i forhold til hinanden. Dette giver derfor:

$$\Delta H = (3 \cdot H(CO_2(g)) + 4 \cdot H(H_2O(g))) - (H(C_3H_8(g)) + 5 \cdot H(O_2(g))) \tag{3.9}$$

Enthalpien indsættes og ændringen beregnes:

$$\Delta H = (3 \cdot (-393,51)kJ + 4 \cdot (-248,80kJ)) - (-103,85kJ + 5 \cdot 0kJ) = -2071,88kJ \tag{3.10}$$

Dermed er ændringen af enthapli af forbrænding af propan $\Delta H = -2071,88kJ$, det vil sige reaktionen er exoterm og afgiver varme med omgiverlserne. Dette passer med virkeligheden, idet de fleste har nok kender til et gasfyr, der brænder gas og dermed opvarmer huset.

- **Opgave 3.6.1:**
Hvad er ændringen i enthapli for dannelsen af NaCl (køkkensalt)?
Reaktionen er $Na^+(aq) + Cl^-(aq) \rightarrow NaCl(s)$
Enthalpien for $H(Cl^-(aq)) = -167,16kJ$, $H(Na^+(aq)) = -240,12kJ$ og $H(NaCl(s)) = -411,15kJ$

- **Opgave 3.6.2:**
Hvad er ændringen i enthalpi for reaktionen $3H_2(g) + N_2(g) \rightarrow 2NH_3(g)$? Enthalpien for molekylerne er $H(H_2(g) = 0$, $H(N_2(g) = 0kJ$ og $H(NH_3(g) = 264,0kJ$

- **Opgave 3.6.3:**
Bestem ændring i enthalpi for kondenseringen af enthansyre (CH_3COOH).
Reaktionen er $2CH_3COOH(l) \rightarrow (CH_3CO)_2O(l)+H_2O(l)$. Der er givet $H(CH_3COOH(g)) = -200,66kJ$, $H(CH_3COOH(l)) = -277,69kJ$, $H(CH_3CO)_2O(l)) = -9,86kJ$, $H(H_2O(g)) = -241,82kJ$ og $H(H_2O(l)) = -285,83kJ$.

- **Opgave 3.6.4:**
Kviksølv oprenses ved en proces kaldet ristning her omdannes kvivsølvsulfid ($HgS(s)$) til rent kviksøl ($Hg(l)$). For dette sker tilsættes oxygen ($O_2(g)$), hvilket giver biproduktet sulfiddioxid ($SO_2(g)$).

  Opskriv og afstem reaktionsligningen. Dernæst beregn ændring i enthalpien og tilstid vurder om reaktionen er endoterm eller exoterm.

$$C_3H_8(g) + 5O_2(g) \rightarrow 3CO_2(g) + 4H_2O(g)$$

Der er givet enthalpierne for hvert molekyle $H(HgS(s) = -53,60kJ$, $H(Hg(l)) = 0kJ$, $H(O_2(g)) = 0kJ$ og $H(SO_2(g)) = -296,83kJ$.

Entropi

Entropi beskriver graden af orden. En analogi til dagligdagen kan være et klædeskab. Hvis alt tøjet er lagt pænt sammen er der orden, og hvis alt er smidt ind i skabbet og delvist udenfor skabbet er der uorden. Ligeledes gælder det for atomer og molekyler. Hvis stoffet er krystallinsk vil atomerne sidde i en gitterstruktur, hvor det ikke er muligt at bevæge sig i. Her er der høj orden. Derimod hvis stoffet er en væske vil molekylerne kunne bevæge sig i forhold til hinanden, hvilket giver mindre orden. Når stoffet kommer i gasfase vil der være den største uorden, idet molekylerne har mere afstand mellem sig end i en væskefase. Et system vil altid gå i mod højest ligesom universet går mod uorden ved konstant udvidelse.

Entropi har symbolet S og når der sker en ændring under en reaktion skrives det som ΔS. Entropi har enheden joule pr. Kelvin ($\frac{J}{K}$). Lad mærke til der er en faktor 1000 mellem energien for entropi (J) og entalpi (KJ). Derudover er entropi temperaturafhængig, idet ved tilføjelse af energi i form af varme, hvilket giver molekylerne energi som som kan løsrive sig en gitterstruktur og blive til væske eller gå fra væske til gas. Temperaturen er i Kelvin og ikke Celsius, som der bruges i dagligedagen. Kelvin skalaen tager udgangspunkt i det absolutte nulpunkt, hvor der ingen molekylevibrationer er og alle molekyler er i fastform og i deres grundtilstand (det rene stof). Ved denne temperatur er der 0 K. Hvis der sammenlignes med Celsius skalaen, så er 0 °C svarende til 273,15 K. Der gælder for alle stoffer, at det fase rene stof har en entropi på $0\frac{J}{K}$.

Regneeksempel med entropi

For at berenge ændringen i entropi for en reaktion anvendes følgende formel:

$$\Delta S = \Sigma(S_{produkter}) - \Sigma(S_{reaktanter}) \tag{3.11}$$

Det vil sige det er alle produkters entropi lagt sammen minium alle reaktanterens entropi lagt sammen. Der ses på en forbrænding af propan (C_3H_8) og oxygen (O_2) til kuldioxid (CO_2) og vand (H_2O):

Som det første findes entropiværdierne for alle stofferne i reaktionen. Dette kan ses i tabellen nedenfor. Hermed er der alle værdier for at kunne beregne ændringen i entropien

Molekyle	Δ S
C_3H_8 (g)	269,91 $\frac{\text{J}}{\text{K}}$
O_2 (g)	205,14 $\frac{\text{J}}{\text{K}}$
CO_2 (g)	213,74 $\frac{\text{J}}{\text{K}}$
H_2O (g)	188,83 $\frac{\text{J}}{\text{K}}$

Tabel 3.2: Entropi værdier for forbrænding af propan

over reaktionen. Her er det støkiometrisk forhold en propan, 5 oxygen, 2 kuldioxid og 4 vand.

$$\Delta S = (3 \cdot S(CO_2(g)) + 4 \cdot S(H_2O(g))) - (S(C_3H_8(g)) + 5 \cdot S(O_2(g))) \tag{3.12}$$

Enthalpien indsættes og ændringen beregnes:

$$\Delta S = \left(3 \cdot 213{,}74\,\frac{\text{J}}{\text{K}} \cdot + 4 \cdot 188{,}83\,\frac{\text{J}}{\text{K}}\right) - \left(269{,}91\,\frac{\text{J}}{\text{K}} + 5 \cdot 205{,}14\,\frac{\text{J}}{\text{K}}\right) = 100{,}93\,\frac{\text{J}}{\text{K}} \tag{3.13}$$

$$C_3H_8(g) + 5O_2(g) \rightarrow 3CO_2(g) + 4H_2O(g)$$

Dermed er ændringen af enthapli af forbrænding af propan $\Delta S = 100{,}93\ \frac{\text{J}}{\text{K}}$. Dette passer med der er en større uorden efter reaktionen end før. Dette skyldes et større antal molekyler.

- **Opgave 3.6.5:**
  Hvad er ændringen i entropi for dannelsen af NaCl (køkkensalt)?
  Reaktionen er $Na^+(aq) + Cl^-(aq) \rightarrow NaCl(s)$
  Entropien er $S(Cl^-(aq)) = 54{,}5\ \frac{\text{J}}{\text{K}}$, $S(Na^+(aq)) = 59{,}0\ \frac{\text{J}}{\text{K}}$ og $S(NaCl(s)) = 72{,}13\ \frac{\text{J}}{\text{K}}$.

- **Opgave 3.6.6:**
  Hvad er ændringen i entropien for reaktionen $3H_2(g) + N_2(g) \rightarrow 2NH_3(g)$?
  Entropien er $S(H_2(g)) = 160{,}7\ \frac{\text{J}}{\text{K}}$, $S(N_2(g)) = 191{,}61\ \frac{\text{J}}{\text{K}}$ og $S(NH_3(g)) = 140{,}6\ \frac{\text{J}}{\text{K}}$.

- **Opgave 3.6.7:**
  Kviksølv oprenses ved en proces kaldet ristning her omdannes kvivsølvsulfid ($HgS(s)$) til rent kviksøl ($Hg(l)$). For dette sker tilsættes oxygen ($O_2(g)$), hvilket giver biproduktet sulfiddioxid ($SO_2(g)$).
  Opskriv og afstem reaktionsligningen. Dernæst beregn ændring i entropien
  Der er givet entropierne for hvert molekyle er $S(HgS(s)) = 88{,}30\ \frac{\text{J}}{\text{K}}$, $S(Hg(l)) = 76{,}02\ \frac{\text{J}}{\text{K}}$, $S(O_2(g)) = 205{,}14\ \frac{\text{J}}{\text{K}}$ og $S(SO_2(g)) = 248{,}22\ \frac{\text{J}}{\text{K}}$.

Gibbs energi

Hverken ud fra enthalpi eller entropi kan der konkluderes om en reaktion vil kunne forløbe eller ej. Det er her Gibbs energien kommer ind i billedet og kombinere de to udtryk og danner et. Gibbs energien har symbolet G, og ved en ændring er det ΔG.

$$\Delta G = \Delta H - T \cdot \Delta S \tag{3.14}$$

Idet entropien afhænger af temperaturen (T) for reaktionen, skal denne ganges på. For denne sammenhæng i ligning 3.14 skal tryk og temperatur være konstant under reaktionen. Reaktionen er ikke spontant, hvis der skal tilføres energi til system konstant. Detter er tilfældet, når Gibbs energien er større end 0.

$$\Delta G > 0$$

Derimod hvis Gibbs energien er mindre end 0, så er reaktionen spontan. Her vil der blive dannet energi ved reaktionen forløber. Dette vil skyldes, at produktet har et lavere energiniveau end reaktanterne.

$$\Delta G < 0$$

Regneeksempel med Gibbsenergi 1

For at berenge ændringen i Gibbs energi kan den beregnes på to metoder. Den første vises i dette eksempel og den anden visen i ekspmpel 2. Den anvendte metode er samme som at finde enthalpiændringen og entropiændringen.

$$\Delta G = \Sigma(G_{produkter}) - \Sigma(G_{reaktanter}) \tag{3.15}$$

Det vil sige det er alle produkters Gibbsenergi lagt sammen minium alle reaktanterens Gibbsenergi lagt sammen. Der ses på en forbrænding af propan (C_3H_8) og oxygen (O_2) til kuldioxid (CO_2) og vand (H_2O):

Molekyle	Δ G
C_3H_8 (g)	$-23,49kJ$
O_2 (g)	$0kJ$
CO_2 (g)	$-394,36kJ$
H_2O (g)	$-228,57kJ$

Tabel 3.3: Gibbsenergiværdier for forbrænding af propan

Som det første findes Gibbsenergiværdierne for alle stofferne i reaktionen. Dette kan ses i tabellen nedenfor. Hermed er der alle værdier for at kunne beregne ændringen i Gibbsenergi over reaktionen. Her er det støkiometrisk forhold en propan, 5 oxygen, 2 kuldioxid og 4 vand.

$$\Delta G = (3 \cdot G(CO_2(g)) + 4 \cdot G(H_2O(g))) - (G(C_3H_8(g)) + 5 \cdot G(O_2(g))) \tag{3.16}$$

Gibbsenergien indsættes og ændringen beregnes:

$$\Delta G = (3 \cdot (-394,36kJ) + 4 \cdot (-228,57kJ)) - (-23,49kJ + 5 \cdot 0kJ) = -2073,87kJ \tag{3.17}$$

Dermed er ændringen af Gibbsenergi af forbrænding af propan $\Delta G = -2073,87kJ$. Det vil sige at reaktionen forløber spontant og molekylerne har et lavere energiniveau efterreaktionen end før.

Regneeksempel med Gibbsenergi 2

Første eksempel var ud fra temperaturen er rumtemperatur (20°C) eller på Kelvinskalaen 298°K. Men reaktionen kan også forløbe ved andre temperaturer her anvendes formlen 3.14.
Metoden er først at bestemme ændringen i enthalpi og dernæst i entropi. Der bliver fortsat set på eksemplet med forbrænding af pentan. Fra regneeksemplerne er $\Delta H = -2071,88kJ$ og $\Delta S = 100{,}93\ \frac{\mathrm{J}}{\mathrm{K}}$ temperaturen sættes til de 298°K.

$$\Delta G = -2071,88kJ - 298\,\mathrm{K} \cdot 100{,}93\,\frac{\mathrm{J}}{\mathrm{K}} \cdot 10^{-3}\,\frac{J}{kJ} = -2101,96kJ \tag{3.18}$$

Der er lidt forskel mellem resultatet fra regneeksempel 1 og 2, hvilket skyldes afrundinger i beregningerne for ændringen i enthalpi og entropi.
Men hvad sker der så ved en forøgelse i temperatur på 500°K for Gibbsenergi?

$$\Delta G = -2071,88kJ - 798\,\mathrm{K} \cdot 100{,}93\,\frac{\mathrm{J}}{\mathrm{K}} \cdot 10^{-3}\,\frac{J}{kJ} = -2152,42kJ \tag{3.19}$$

Her kan det ses Gibbsenergi bliver mere negativ, men faktisk ikke særlig meget i forhold til det er en temperaturændring på 500°K.

- **Opgave 3.6.8:**
  Hvad er ændringen i Gibbsenergien for reaktionen af spaltning af hydrogenperoxid (H_2O_2) $2H_2O_2(aq) \rightarrow 2H_2O(l) + O_2(g)$? Gibbsenergi for molekylerne er $G(H_2O_2(aq) = 120,35kJ$, $G(H_2O(l) = -237,13kJ$ og $G(O_2(g)) = 0kJ$.
  Bonus spørgsmål en katalysator mindsker energien, der er nødvendig for en reaktion forløber. Hvordan vil en katalysator påvirke denne reaktion?

- **Opgave 3.6.9:**
  Til dannelsen af Magnesiumoxid (MgO) er enthalpien $\Delta H(MgO(s)) = -601,70kJ$ og entrophien $\Delta S(MgO(s)) = 26{,}94\ \frac{\mathrm{J}}{\mathrm{K}}$. Bestem Gibbsenergien ved følgende temperaturer:

  - 100K

Molekyle	ΔH	ΔS	ΔG
$H_2O(g)$	-241,82kJ	188,83 $\frac{J}{K}$	-228,57kJ
$H_2O(l)$	-285,83kJ	69,91 $\frac{J}{K}$	-237,13kJ
$CO_2(g)$	-393,51kJ	213,74 $\frac{J}{K}$	-394,36kJ
$CO_2(aq)$	-413,80kJ	117,6 $\frac{J}{K}$	-385,98kJ
$H_2CO_3(aq)$	-699,65kJ	187,4 $\frac{J}{K}$	-623,08kJ

Tabel 3.4: Enthalpi- og entropiværdier

- 298K
- 750K
- 5000K

●● **Opgave 3.6.10:**
Hvad er ændringen i Gibbsenergi for dannelsen af BaCl? Er reaktionen spontant?
Reaktionen er $Ba^{2+}(aq) + 2Cl^{-}(aq) \rightarrow Ba_2Cl(s)$
Gibbsenergien for de enkelte molekyler er $G(Cl^{-}(aq)) = -131,23kJ$, $G(Ba^{+}(aq)) = -560,77kJ$ og $G(Ba_2Cl(s)) = -810,40kJ$.

●● **Opgave 3.6.11:**
Hvad er ændring i Gibbsenergi for dannelsen af eddikesyre? Der er givet enthalpien er $\Delta H(CH_3COOH(aq)) = -485,76kJ$ og entrophien $\Delta S(CH_3COOH(aq)) = 178{,}70\ \frac{J}{K}$ og temperaturene:

- -75°C
- 0°C
- 20°C
- 55°C
- 2750°C

●●● **Opgave 3.6.12:**
Kuldioxid ($CO_2(aq)$) kan reagere med vand ($H_2O(l)$) og danne en syre kaldet kulsure ($H_2CO_3(aq)$). I tabellen er givet forskellige enthalpier og entropier.

1) Opstil reaktionen

2) Bestem enthalpi og entropi samt Gibbsenergi for reaktionen ud fra tabel3.4

3) Bestem Gibbsenergien for 0°C, 100°C og 1000°C

Idealgas ligningen

En anden del af termodynamikken er at se på en gas og dets opførsel. For at kunne lave matematiske modeller af virkeligheden er det ofte nødvendig at simplificere og lave nogle antagelser. For gasser er det den perfekte gas. Denne gas påvirker gasmolekylerne ikke hinanden, hvilket gør man ikke skal tage højde for deres interne frastødning eller tiltrækning af hinanden. Med denne antagelse er idealgas ligningen lavet.

$$p \cdot V = n \cdot R \cdot T \tag{3.20}$$

Her trykket p og i kemi har den ofte enheden bar. V er volumenet og er angives i L altså liter. Stofmængden n har enheden mol. Temperaturen T er i Kelvin K. R er en konstant kaldet gaskonstanten og har værdien $R = 8{,}3145\ \frac{bar \cdot L}{molK}$. Hvis der ses på konstanterne og

enhederne på højresiden af idealgas ligningen (ligning 3.20) kan det ses enhederne for temperatur og stofmængde er i nævneren på idealgas ligningens enhed, hvilket gør de gå ud mod hinanden. Dermed er der enhederne for volumen og tryk tilbage som også er på venstresiden af ligningen.
Med idealgas ligningen kan der bestemmes en parameter for et system ud fra de andre. Eksempelvis kan trykket findes, hvis volumenet, stofmængden og temperaturen er kendt. Her vil der isoleres for p, ved at dividere med V. Dette giver så:

$$p = \frac{n \cdot R \cdot T}{V} \tag{3.21}$$

3.7 Titrering af Fe(II) med $KMnO_4$

Indledning

I kemien er det tit nødvendigt at bestemme stofmængden eller koncentrationen i en ukendt prøve. En af de analysemetoder der bruges er en titrering. I denne titrering skal i bestemme koncentrationen af Fe(II) i en opløsning. Dette kan f.eks. være relevant hvis man skal bestemme mængden af Fe(II) i den jernkatalysator der benyttes industrielt når man skal producere ammoniak. Ammoniak bruges både til gødning og til bomber.

Figur 3.12: Billede af $KMnO_4$ salt kilde([**Wikimedia:Potassiumpermanganate**])

I titreringen benyttes der $KMnO_4$ (se 3.12) som *reagens* og som *indikator*. Det er $KMnO_4$ der reagerer med Fe(II)'en, og også den der sørger for at opløsningen skifter farve når al Fe(II)'en i den ukendte opløsning er reageret med. Dette tidspunkt kalder man ækvivalenspunktet - det punkt hvor man har tilsat lige så meget $KMnO_4$ som der er Fe(II) i opløsningen. Stoffet er også indikator, da det vil føre til at opløsningen skifter farve ved ækvivalenspunktet.

Følgende reaktion sker i kolben når der tilsættes $KMnO_4$:

$$5\,Fe^{2+} + MnO_4^- + 8\,H^+ \longrightarrow Mn^{2+} + 5\,Fe^{3+} + 4\,H_2O \qquad \{7\}$$

MnO_4^- er en stærkt farvet ion (pink), men hvis der stadig er Fe^{2+} tilbage i opløsningen reagerer ionerne og bliver til de farveløse Mn^{2+}. Her er det vigtigt at huske at for hver 5 Fe(II)-ioner reagerer kun 1 MnO_4^-. Dvs. ved ækvivalenspunktet hvor der er **en blivende pink farve** gælder ligning 3.22:

$$n(Fe^{2+}) = 5 \cdot n(MnO_4^-) \tag{3.22}$$

Denne skal i bruge til at beregne stofmængden i den ukendte opløsning.

Formål

Formålet med øvelsen er at betsemme koncentrationen af Fe(II)-ioner i den ukendte opløsning.

Forsøgsvejledning

Til titreringen vil der blive brugt $KMnO_4$ som titrand, hvor reaktionen 7 vil ske. Når der er tilsat en ækvivalent mængde MnO_4^--ioner til den ukendte opløsning vil der være en blivende pink farve.

Sikkerhed

I laboratoriet skal der **altid** bruges sikkerhedsbriller, langt hår skal være sat op, og man skal have **lukket** kittel på. Når man forlader laboratoriet, skal man **altid** vaske hænder før man går ud. Hvis man spilder kemikalie på hænderne skal man skylle med rigeligt vand. Før øvelsen starter, bliver der gennemgået sikkerhedsregler dybere.

Kemikalieaffald skal som tommel-fingerregel opsamles i kemikaliedunke. I denne øvelse laves der dog blot Fe^{3+}, Mn^{2+} og vand. Disse kan alle hældes ud i vasken. Er der derimod stadig MnO_4^--ioner tilbage, må det **ikke** hældes ud, da permanganationer forurener havmiljøet. Det vil sige at **hvis man har titrand i overskud må det ikke komme ud i vasken**.

Materialer

I listen kan der ses alle de materialer der skal bruges til forsøget:

- Kemikalier og opløsninger
  - 0,02 M $KMnO_4$-opløsning
  - 25 mL ukendt $FeSO_4$-opløsning
- Glasudstyr
  - 25 ml burette
  - To 50 ml bægerglas
  - 50 ml konisk kolbe
  - 25 mL glaspipette
- Resterende udstyr
  - A-fodsstativ
  - Buretteholder
  - Pipettebold
  - Engangspipetter
  - Tragt

Metode

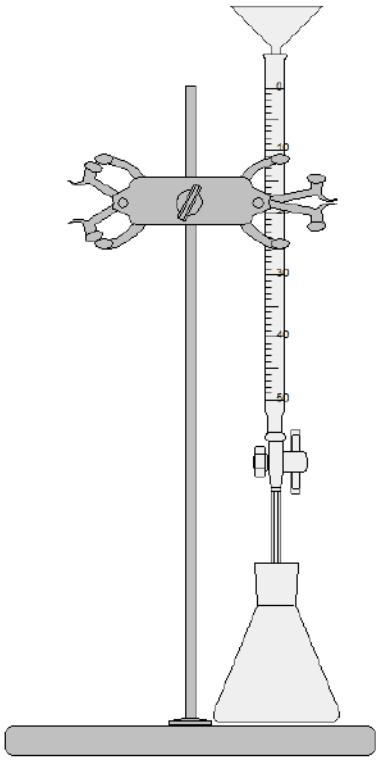
Sæt buretten fast til stativet ved at bruge buretteholderen, sørg for at buretten er placeret lodret og at hanen er lukket. Tragten sættes i toppen af buretten og kolben stilles under. Se 3.22 for billede af opstilling. Når du hælder titrand ($KMnO_4$) op skal der stilles et tomt bægerglas under buretten.

Overfør ved hjælp af en fuldpipette 25 ml af den ukendte opløsning (prøven) af $FeSO_4$ til en 50 ml konisk kolbe. Overfør ca. 25 ml af 0,02 M $KMnO_4$ til et 50 ml bægerglas. Tag begge kemikalier tilbage til opstillingen af forsøget.

Kontrollér at hanen på buretten er lukket og derefter fyld buretten op med 0,02 M $KMnO_4$. Placér et 50 ml bægerglas under buretten, åbn for buretten og lad et par milliliter løbe igennem, luk derefter for buretten. Dette er for at at fjerne luftboblen i bunden af buretten. Aflæs så startvolumen på buretten og skriv den ned.

Placér den koniske kolbe med prøven under buretten. Start titreringen ved at åbne stille og roligt for buretten. Der titreres optimalt ved at swirle den koniske kolbe en gang

imellem så væsken blandes helt. Fortsæt indtil, at der opnås et permanent farveskifte til pink, hvor ækvivalenspunktet da er nået. Når ækvivalenspunktet er nået, noteres den nuværende volumen af buretten. **Det er det brugte volumen 0,02 M $KMnO_4$ der skal bruges til at beregne koncentrationen af den ukendte $FeSO_4$-opløsning**.



Figur 3.13: Figur af titreringspostilling med tragt, burette i stativ, og kolbe.

Resultatbehandling

Notér først det tilsatte volumen af NaOH i tabellen og beregn derefter koncentrationen af NaOH.

	Titrering 1	Titrering 2	Titrering 3
$V(\mathrm{KMnO_4})_{start}$			
$V(\mathrm{KMnO_4})_{slut}$			
$V(\mathrm{KMnO_4})_{tilsat}$			
$n(\mathrm{KMnO_4})$			
$n(\mathrm{Fe^{2+}})$			
$c(\mathrm{Fe^{2+}})$			

Tabel 3.5: Tabel til notering af tilsat volumen og beregning af koncentration ved titrering.

3.8 Endoterm reaktion mellem citronsyre og bagepulver

Indledning

Køkkenkemi er ikke noget rigtigt begreb der beskriver en gren af kemien. Den er blot den kemi man i princippet kan lave derhjemme i køkkenet, og sådan kemi er tit god til at demonstrere svære koncepter. I dette forsøg skal i lave en endoterm reaktion (en reaktion der forbruger varme, og derfor bliver kold) mellem to køkkenkemikalier, nemlig citronsyre ($\mathrm{C_6H_8O_7}$) og bagepulver (primært $\mathrm{NaHCO_3}$). Reaktionen mellem disse i vand er en endoterm syre-base reaktion:

$$\mathrm{NaHCO_3 + C_6H_8O_7 \longrightarrow Na^+ + CO_2 + H_2O + C_6H_7O_7^-} \quad \{8\}$$

Her er citronsyren en syre, og bagepulveret en base. Reaktionen danner $\mathrm{CO_2}$-gas. Derved er entropien af reaktionen positiv, og går op for at entalpien af reaktionen også er positiv. Husk på er Gibbs-energien (ΔG) skal være negativ i ligningen:

$$\Delta G = \Delta H - T \cdot \Delta S \tag{3.23}$$

for at reaktionen sker. Vi siger derfor at der er *entropien der driver reaktionen.*

Formål

Formålet med forsøget er at demonstrer en endoterm reaktion (en reaktion der *forbruger* varme).

Forsøgsvejledning

Sikkerhed

I laboratoriet skal der **altid** bruges sikkerhedsbriller, langt hår skal være sat op, og man skal have **lukket** kittel på. Når man forlader laboratoriet, skal man **altid** vaske hænder før man går ud. I forsøget bruges der koncentreret citron syre (som er en middelstærk syre). Hvis man spilder kemikalie på hænderne skal man skylle med rigeligt vand. Bagepulver er ufarligt. Kemikalieaffald skal blot i vasken. Før øvelsen starter, bliver der gennemgået sikkerhedsregler dybere.

Materialer

I listen kan der ses alle de materialer der skal bruges til forsøget:

- Kemikalier
  - 1 teske citronsyre
  - 1 teske bagepulver
- Glasudstyr
  - 50 ml konisk kolbe
- Resterende udstyr
  - Engangspipette

Metode

1 teske citronsyre og 1 teske bagepulver blandes i en 50 mL konisk kolbe. For at reaktionen sker tilsættes der en halv pipette vand. Kolben vil nu føles kold på bunden da reaktionen forbruger varmen.

3.9 Undersøgelse af reaktioner er endoterme eller exoterme

Indledning

Reaktioner kan være endoterme eller exoterme. Ved at undersøge, hvilken type det er, kan man sige hvilket fortegn enthalpien har for reaktionen. I forsøget skal der undersøges om nogle reaktioner er endoterme eller exoterme, når de forløber. Den første reaktion er med calciumixid (CaO) og vand.

$$\mathrm{CaO(s) \equiv H_2O(l) \longrightarrow Ca(OH)_2(aq)} \qquad \{9\}$$

Den næste reaktion er med ammoniumnitrat ($NH_4NO_3(s)$) og vand:

$$\mathrm{NH_4NO_3(s) + H_2O(l) \longrightarrow {NH_4}^{+}(aq) + {NO_3}^{1-}(aq) + H_2O(l)} \qquad \{10\}$$

Den tredje reaktion er med bariumchlorid ($BaCl_2$) i vand.

$$\mathrm{BaCl_2(s) + H_2O(l) \longrightarrow {Ba}^{2+}(aq) + 2\,{Cl}^{1-}(aq) + H_2O(l)} \qquad \{11\}$$

Formål

Formålet ved forsøget er at se bestemme om en reaktion er endoterm eller exoterm ud fra dets varmeudvikling. Der skal undersøges om reaktionen forbruger varme fra omgiverlserne eller afgiver varme til omgivelserne.

Sikkerhed

I laboratoriet skal der **altid** bruges sikkerhedsbriller, langt hår skal være sat op, og man skal have **lukket** kittel på. Når man forlader laboratoriet, skal man **altid** vaske hænder før man går ud. I forsøget bruges der koncentreret citron syre (som er en middelstærk syre). Hvis man spilder kemikalie på hænderne skal man skylle med rigeligt vand. Bariumchorid, calciumoxid og ammoniumnitrat kan give hud- og øjenirriatiation, hvilket gør det er vigtigt at vaske det af ved berøring med stoffet.

Materialer

- Kemikalier
  - 12,5 g calciumoxid, CaO
  - 25 g ammoniumnitrat, NH_4NO3
  - 10 g bariumchlorid, $BaCl_2$
  - Vand
- Glasudstyr
  - 3 stk. 200 mL bægerglas
  - 3 stk. 50 mL bægerglas
- Resterende udstyr
  - Termometer
  - Spatel

Metode

Afvej 12,5 g calciumoxid i 50 mL bægerglasset. Tilsæt 100 mL vand til 200 mL bægerglasset. Sæt termometret i vandet og noter temperaturen inden calciumoxid tilsættes. Når temperaturen er noteret tilsæt calciumoxid og bland. Hold øje med temperaturændringen og noter den ned.
Den samme medtode anvendes til undersøgelsen af ammoniumnitrat med 25 g i 100 mL vand og bariumchlorid med 10 g i 100 mL vand.

Databehandling

Jeres observationer noteres ned i tabel 3.6, hvor der konkluderes derfra om reaktionen var endoterm eller exoterm.

Stof	T_{start}	T_{slut}	$\Delta T = T_{slut} - T_{start}$	Endoterm	Exoterm
Calciumoxid					
Ammoniumnitrat					
Bariumchlorid					

Tabel 3.6: Ændring af temperatur og konklusion af reaktionstype

3.10 Ekstra opgaver

Opgave 3.10.1: Kjeldahlanalyse
De fleste planter man dyrker på industrielt niveau får gødning. I gødning er der bl.a. nitrater (NO_3^-) som planten bruger som nitrogenkilde. Nitrogen indgår bl.a. i plantecellernes DNA, RNA og aminosyre og er derfor nødvendig for, at planten kan lave celledeling og dermed gro. Klorofyl, som giver planter en grøn farve, indeholder nitrogen. Så hvis planten ikke har nok nitrogen bliver bladene gullige. Det er derfor meget vigtigt at vide hvor meget nitrat, man har i ens gødning.

Til dette bruger man en Kjeldahl analyse, som kan analysere indholdet af nitrogen i enhver prøve. Nitraten omdannes til ammoniak, som der titreres på. Dette sker ved at bruge en aluminiumslegering, hvor der sker denne reaktion:

$$3\,NO_3^- + 8\,Al + 5\,OH^- + 18\,H_2O \longrightarrow 8\,[Al(OH)_4]^- + 3\,NH_3$$

Titratoren er en NaOH-opløsning lavet ud fra at opløse 25 g NaOH salt i 100 ml demineraliseret vand.

1) Beregn koncentrationen af titratoren

Der bliver opløst 10 g gødning i 100 ml vand, der overføres til en 150 ml konisk kolbe. Der titreres derefter med NaOH og ækvivalenspunkt nås ved brug af 47 ml titrator.

2) Beregn stofmængden af brugt NaOH

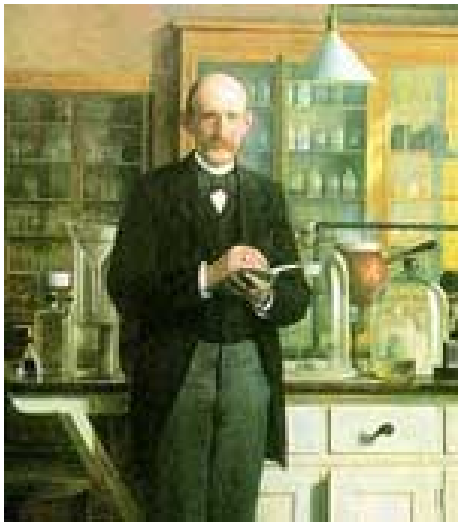
3) Beregn stofmængden af nitrat

4) Beregn koncentration af nitrat i den 50 ml opløsning af gødning

Aluminium bliver tilsat i form af devardas legering, legeringen består af 45% Al, 50% Cu og 5% Zn. Procenterne er i masseprocenter, det vil sige at i et 1g devardslegering er der 0,45g Al, 0,5g Cu og 0,05g Zn.

5) Beregn hvor mange gram devards der som minimum skal bruges til titreringen.

Kjeldahl analyse andvendes stadigvæk i høj grad i industrien, Kjeldahl udviklede metoden, mens han arbejdede på Carlsberglaboratoriet, et portræt af ham kan ses på figur 3.14.



Figur 3.14: Maleri af den danske Johan Kjeldahl af Otto Haslund

For at kende indholdet af nitrit udføres spektrofotometrisk analyse på en opløsning med 10 g gødning i 100 ml vand. Aborsorbansen er blevet målt til 0,34 og absorptions koefficient for den farve produkt er i denne opgave 27 $\frac{\mathrm{L}}{\mathrm{mol}}$.

6) Beregn koncentration af nitrit i opløsning.

Det er et krav fra landmanden at ud af den samlede mængde af NO_2^- og NO_3^- må max 5% af den være nitrit

7) Overholder gødningen kriteriet fra landmanden?

Opgave 3.10.2: Fritz Haber
Den tyske videnskabsmand Fritz Haber og kemiingeniøren Carl Bosch udviklede Haber–Bosch processen (ligning (12)) i starten af 1900-tallet, dette gjorde at man ikke

længere var afhængig af ekskrementer fra dyr for at få ammoniak. I dag er Haber–Bosch processen meget brugt, da den er meget vigtigt for produktionen af gødning.

$$N_2 + 3\,H_2 \longrightarrow 2\,NH_3 \qquad \{12\}$$

Tyskerne brugte blandt ammoniak til at fremstille salpetersyre HNO_3, Der ønskes at fremstilles 2 mol ammoniak.

1) Kan det fremstilles ud fra 2,1 mol N_2 og 2 mol H_2 når det antages, at reaktion forløber 100% ?

2) Hvor mange mol N_2 og H_2 bruge for at fremstille 15 mol NH_3

Salpetersyre kan blive syntetiseret ud fra en proces kaldet Ostwald processen.

$$NH_3(g) + 2\,O_2(g) \longrightarrow H_2O(g) + HNO_3(aq)$$

3) Hvor mange mol HNO_3 bliver der dannet ud fra 15 mol NH_3, når der er et overskud af ilt og hvor mange mol O_2 bliver der brugt?

4) Hvor mange ml 17 M HNO_3 bliver der fremstillet ud fra 15 mol NH_3?

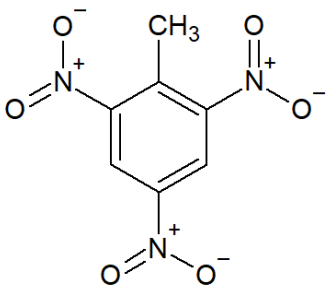
For at tjekke efter om produktet har en koncentration er 17 M, bruges der en pH analyse. Der bliver udtaget 10 ml af produktet, som bliver hældt over i en 100 ml målekolbe med 67 ml vand.
Derefter bliver der til føjet med vand op til kanten. Den fortyndet opløsning bliver målt til at have en pH på -0,23.

5) Hvad er koncentrationen af den fortyndet opløsning og er koncentration af produktet 17 M HNO_3?

En anden grund til at Haber-Bosh processen var meget vigtigt for tyskerene var at i starten af 1900-tallet foregik første verdenskrig, hvor man har brug for ammoniak for at kunne fremstille ammunition.

$$C_7H_8 + 3\,HNO_3 + 3\,H_2SO_4 \longrightarrow C_7H_5(NO_2)_3 + 3\,HSO_4^- + 3\,H_3O^+ \qquad \{13\}$$

6) Hvor mange kg TNT($C_7H_5(NO_2)_3$) kan der fremstilles ud fra 4,1 kg toluen (C_7H_8)), når der er et overskud af salpetersyre (HNO_3) og svovlsyre (H_2SO_4)



2-methyl-1,3,5-trinitrobenzen

Figur 3.15: Sturktur af TNT

Opgave 3.10.3: Titrering
En opløsning af 50 ml eddikesyre tilsættes NaOH. Når der er tilsat 18 mL 2 M NaOH, er der lige meget syre og base i blandingen. Hvad er stofmængden af eddikesyre i den originale blanding?
Hint:
Eddikesyre og natriumhydroxid reagerer efter følgende reaktionsskema

$$CH_3COOH + NaOH \longrightarrow CH_3COONa + H_2O \quad \{14\}$$

Der bliver foretaget en pH-måling af 10 ml eddikesyre fortyndet med 40 ml vand og pH-målingengiver et resultat på pH=2,8.

1) Stemmer det overens med hvad man ville forvente når pKs af eddikesyre er på 4,76?

Kapitel 4

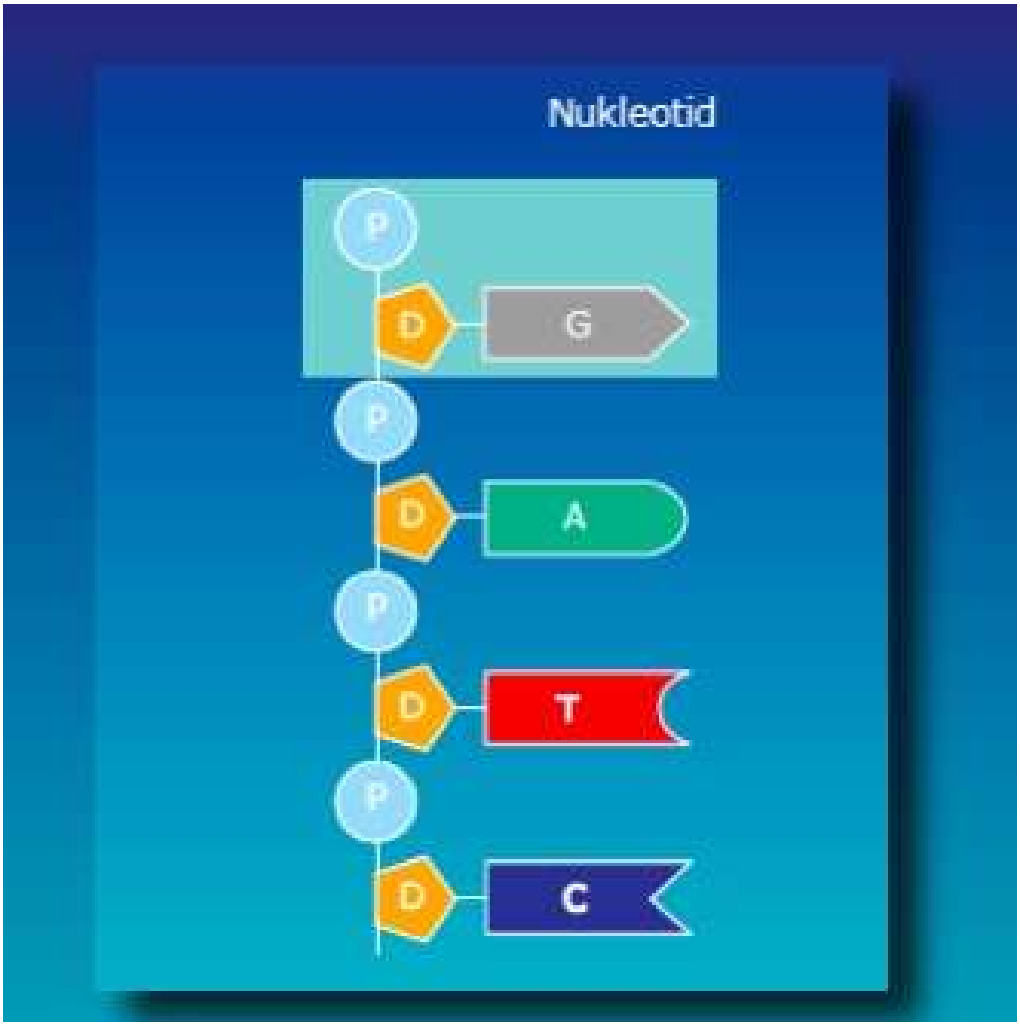
Biologi

4.1 Introduktion

Biologi er kort fortalt studiet af alt levende. Biologer undersøger alt mellem hvordan hele arter lever med verden omkring den til hvordan en enkelt organel bliver reguleret. Dette er selvfølgeligt lige rigeligt at dække her, det kunne faktisk godt tage et par menneskealdre at sætte sig ind i det hele, så i stedet vil vi starte med noget, som er helt centralt for liv. DNA. DNA, eller rettere arvemateriale, er måden hvorpå information bliver videregivet mellem generationer. Det er desuden vigtigt, når vi skal gå fra et lille æg til et pattedyr med ben og organer eller en bakterie som kan overleve i 70 °C. I det her kompendium kommer vi til at gennemgå 1. Hvad er DNA, 2. Hvordan replikeres DNA og 3. Hvordan kan vi regulere udtrykningen af DNA? Det sidste forklarer hvorfor vi har arme oppe ved hovedet og ben nede ved jorden i stedet for omvendt.

4.2 DNA, hvad er det og hvordan ser det ud?

DNA er hele opskriften for hvordan vi som menneske er (altså biologisk, personlighed holder vi os på lang afstand af). Det er så at sige en opskrift på, hvordan vi ser ud, de genetisk instruktioner, der bruges, når vi vokser og udvikler os. Men det er ikke kun i mennesket, at DNA findes. Der er også tale om DNA, hvis vi kigger på mange vira, bakterier, svampe osv. Alt levende indeholder arvemateriale, altså DNA. Men hvad er det så? Jo, DNA er først og fremmest en forkortelse, en forkortelse for "Deoxyribo-Nucleic-Acid", hvilket på dansk betyder, at DNA er en kernesyre, da den findes i cellekernen hos eukaryote celler. Syreegenskaben gør, at DNA kan afgive en proton, og dermed blive negativt ladet. Dette er en fordel, fordi DNA så kan binde sig til histoner, som tilsidst munder ud i, at der bliver dannet et færdigt kromosom. DNA består af 6 forskellige byggesten: Kulhydratet deoxyribose (D'et i DNA), Fosfat og de fire baser, som har navnene Adenin (A), Guanin (G), Thymin (T) og Cytosin (C). DNA'ets udseende, er den typiske stigestruktur, hvor baserne udgør stigens trin, mens deoxyribose og fosfat udgør stigens to "ben". De to "ben" er to lange spiraliserende kæder, en såkaldt dobbelthelix. Den enkelte DNA-streng er opbygget af hundredevis af nukleotider, som hver især består af en deoxyribose, hvilket er en bestemt type sukker, en fosfat-gruppe og en nitrogenbase. Sukkerstoffet og fosfatgruppen er den samme, på alle nukleotider, det eneste der varierer er baserne. Figur 4.1 viser et eksempel.



Figur 4.1: Overstående figure viser en enkelt DNA streng med de 4 baser. Rygsøjlen er ens og kun basernes struktur ændres. [**SkadhedeHyldal**]

De 4 baser kan indeles i 2 grupper puriner og pyrimidiner. De to grupper indeholder hver især 2 baser. Adenin (A) og Guanin (G) er de to puriner, mens Thymin (T) og Cytosin (C) er pyrimidiner. De to DNA strenge i dobbelthelix strukturen er bundet sammen via hydrogen bindinger mellem de to baser, dette kaldes baseparring. Hver DNA streng har en rygrad, som består af deoxyribose-fosfat-delen, på denne rygrad sidder nitrogenbaserne, som vender ind mod hinanden mellem de to DNA strenge. Hver base binder kun med en specifik anden af de 3 andre baser, dette kaldes komplementær baseparring, eller specifik baseparring, således at Adenin altid pare med Thymin, og Cytosin altid pare med Guanin. Dette medføre, at der altid er et lige antal Adenin og Thymin, samt et lige antal Cytosin og Guanin.

DNA er det arvelige materiale

I dag virker det naturligt, at det er DNA vi snakker om, når vi snakker arvemateriale, men sådan har det ikke altid været. Tilbage før omkring 1940'erne, troede de fleste faktisk, at det måtte være et protein, der var det arvelige materiale. Det er fordi proteiner var de eneste strukturere, der virkede komplekse og diverse nok til at kunne være arvemateriale. DNA virkede til sammenligning som en relativt simpel, og dårligt forstået, gruppe molekyler, som kom i bestemte ratioer. Det var først rigtig omkring 1950, hvor Hershey-Chase eksperimentet blev fremvist, at DNA blev anerkendt som arvematerialet. Det de startede med var, at radioaktivt markere svovl og fosfor. Derefter tog de en

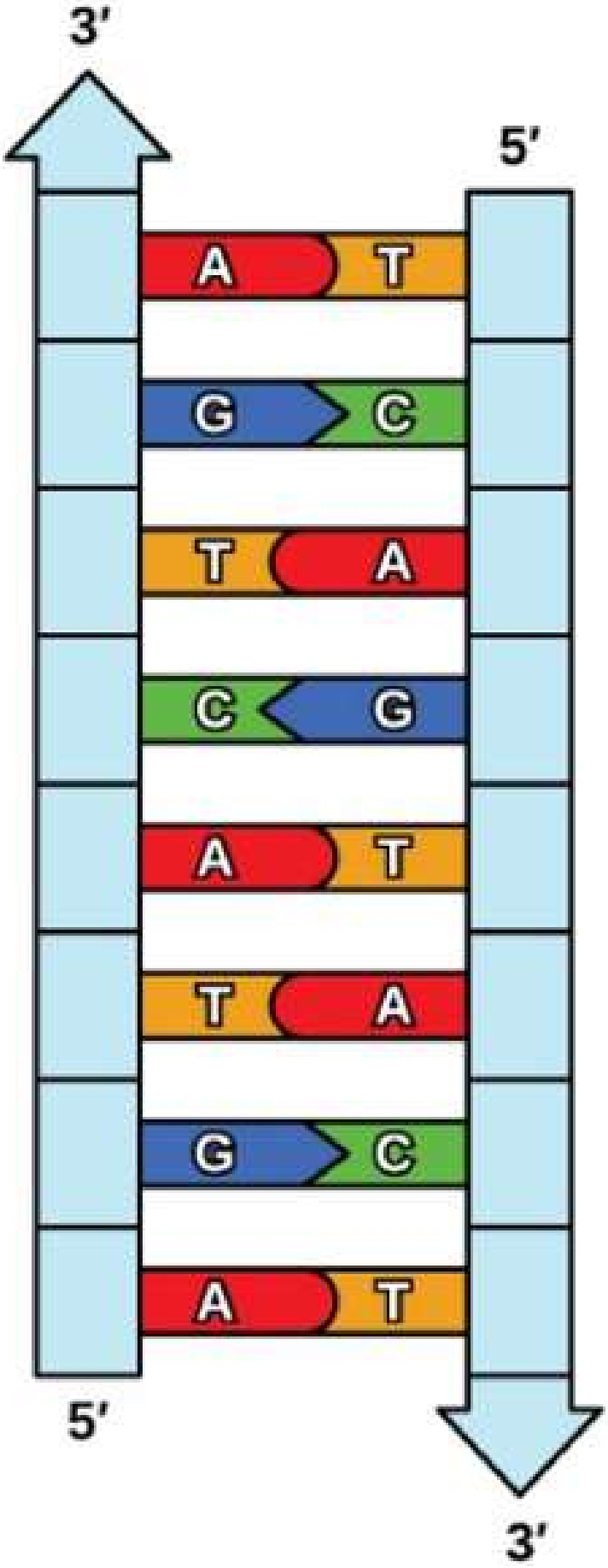
population af bakteriofager (som er vira der angriber bakterier) og groede halvdelen på dem radioaktive svovl og den anden på den radioaktive fosfor. Man vidste allerede, at bakteriofager indfører deres genetiske materiale, og ikke meget mere. De to inficerede bakteriofagekulturer, fik så lov at inficere hver deres bakteriekultur. Man målte radioaktiviteten og så blev kulturerne blendet. Dette skilte bakteriofagen og bakterien fra hinanden , men ødelagde ikke bakterie cellerne. Bagefter målte de radioaktiviteten igen. Det viste sig at de bakterier der var inficerede med svovl markerede bakteriofager mistede 80% af deres signal, hvorimod dem der havde fosformarkering, kun mistede omkring 40% af deres signal. Dette tog man som evidens for, at det var DNA som var det arvelige materiale og ikke protein.

Spørgsmål Kan du foreklare hvorfor det gav mening at bruge svovl og fosfor som markøer?

4.3 DNA og dens replikation

DNA replikation er forudsætningen for en succesfuld celledeling, processen finder sted hos både eukayoter og prokayoter. Formålet med processen er, at kopiere cellens DNA, så en kopi af dette kan videregives til den nye celle under celledeling. Dette er vigtigt da der ellers aldrig ville blive dannet nye celler og alt liv ville uddø ret hurtigt. Processen kan opsummeres i 4 trin.

Trin 1 I dette trin, foldes de to DNA - strenge i dobbelthelix strukturen ud fra hinanden. For at de to DNA - strenge kan skilles fra hinanden, skal bindingerne mellem baseparene, Adenin (A), Thymin (T), Cytosin (C) og Guanin (G) først brydes. Dette gøres af enzymet DNA-helikase. DNA-helikase bryder hydrogenbindingerne mellem baseparene således, at de to DNA -strenge splittes i en y - lignende form, kendt som replikationsgafflen. Man kan fristes til at tro, at en ny streng så vil blive dannet ud fra begge de to gamle, men helt så simpelt er det ikke. Det er nemlig sådan, at DNA er retningsbestemt. Begge DNA strenge er iført en 3´-og 5´ ende. At en DNA streng har en 3´er og 5´er ende angiver blot hvilken funktionel gruppe, der er tilknyttet en af enderne. 5´er enden har en fosfat gruppe bundet, og 3´er enden har en hydroxylgruppe gruppe bundet. Selve replikationen er også retningsbestemt, da replikationen kun kan køre i en 5´er til 3´er retning. Det vil altså sige, at replikationen kører fra strengenes 5´er ende og hen til den modsatte 3´er ende.



Figur 4.2: På ovenstående figur ses de to DNA strenge der er bundet sammen med deres komplimentære basepar, derudover ses også hvilken retning de to DNA strenge vender i forhold til hinanden i forhold til deres 3´er og 5´er retning. billedet er taget fra [7]

Selve replikationsgaflen er tovejs. Det vil sige, at de to DNA strenge, der repliceres, er placeret modsat hinanden, således at 5´er enden på den ene DNA streng er placeret overfor den anden DNA strengs 3´er ende. DNA strengen med en 3´er - 5´er retning kaldes den førende streng, mens strengen med en 5´er til 3´er retning kaldes den bagudrettede streng. DNA vil altid blive dannet i modsat retning til den streng det dannes ud fra. Det betyder, at strengen som går 3' til 5' vil blive repliceret i en 5' til 3' manér. Dette er den vej replikation fungere i og der er ingen problemer, derfor "den førende streng". Problemet opstår med den anden streng, som skal repliceres modsat den retning replikation foregår i. Dette kommer vi snart ind på.

Trin 2 Det er i dette trin at selve replikationen begynder. Til at starte replikationen, dannes et lille stykke RNA kaldet en primer. Primeren dannes af enzymet DNA - primase. Primeren sætter sig på 3´er enden på den førende streng. Primeren danner startpunktet for selve replikationen og den førende streng er som sagt den simpleste at replicere. Mens der blot bindes en enkelt primer på den førende streng, bindes der adskillige primere på den bagudretteede streng, hvor hver primer ligger tæt på hinanden. Det gøres fordi, den bagudrettede streng skal repliceres i små dele, grundet retningen, og det betyder, at man må starte processen flere steder.

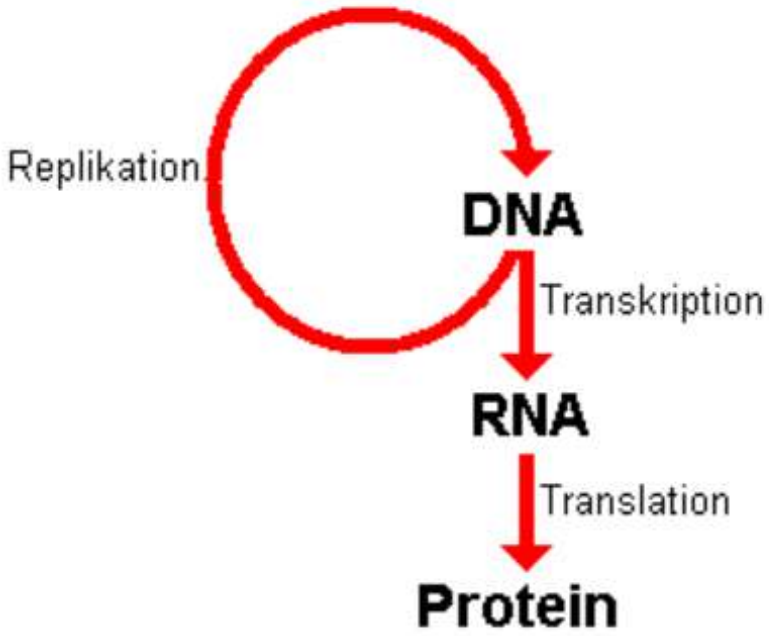
Trin 3 Her dannes nye DNA strenge gennem en process kaldet forlængelse. Den nye streng skabes af DNA - Polymeraser. I alt kendes der 5 forskellige typer for DNA - polymeraser. Polymerase III er det vigtigste replikationssenzym, mens de resterende 4 polymerase I, II, IV og V står for for fejlkontrol og reparation. På den førende DNA streng sætter replikationsenzymet DNA polymerase III sig på det sted hvor primern har sat sig. Herefter kører den fra 5´er enden til 3ér enden, hvor polymerasen sætter de komplimentære basepar over for de allerede sidende basepar på strengen. På den bagudrettede DNA - streng adskilles de nyfragmenterede DNA - strenge, da DNA - polymerasen binder sig til mellemrummende mellem de forskellige primere. DNA polymerase kan altså kun danne nyt DNA i mellemrummet mellem primere. Disse DNA stykker kaldes Okazaki-fragmenter. Da DNA strengen fragmenteres på den bagudrettede DNA - streng kaldes replikations processen for denne diskontinuerlig. Mens den kaldes kontinuerlig for førestrengen, da denne ikke fragmenteres og derved beholdes intakt.

Trin 4 Når de to DNA strenge er dannet, fjernes primerne fra DNA strengene, og erstattes med passende baser, hvorefter den nydannede streng korrektionslæses for eventuelle fejl. Når den nydannede DNA streng er færdigdannet, forbindes Okazaki-fragmenterne, via et enzym kaldet DNA -ligase. Når Okazaki-fragmenterne er blevet samlet, vil de to nye, og ens, DNA strenge blive oprulles til den karakteristiske dobbelthelix streng.

Spørgsmål kan du tænker dig til en anvendelsesmulighed af viden om hvor mange celler der normalt er igang med at dele sig på ethvert givent tidspunkt.

Protein Syntesen

Når nu vi snakker om DNA, så bliver vi næsten nød til at nævne det centrale dogme og med det proteinsyntesen. Figur 4.3 viser processen. Ideen er at information går fra DNA til RNA til protein. Det er siden vist at det ikke er helt rigtigt, men det giver et fint overblik.



Figur 4.3: Ovenstående figur viser trinnene i rækkefølge for DNA replikationen, hvor produktet er et færdigdannet protein. Billedet er taget fra quizlet.

Med det in mente, vil vi nu præsentere proteinsyntesen, en stor grund til at vi overhovedet kan fungere.

Proteinsyntesen er den proces hvor der dannes proteiner, i cellen ud fra dets genetiske materiale (DNA). Proteiner der dannes ud fra protein syntesen, kan enkeltvis dannes til mange forskellige formål, fx. proteiner til opbygning af celledele, signalstoffer, opbygning af muskler mm. Overordnet set kan proteinsyntese opdeles i 2 dele. 1) transkription - DNA aflæses og den genetiske kode oversættes, til dannelsen af mRNA. 2) translation - her aflæses mRNA, hvorefter der dannes et protein i cellens cytosol.

Transkription

Den første del af protein syntesen kaldes også RNA - syntesen. I denne fase af proteinsyntesen, syntetiseres et RNA - molekyle. RNA - molekylet indeholder den genetiske kode for et enkelt gen. Der findes mange typer for RNA molekyler, der hver især tjener forskellige formål i kroppen, men den type af RNA - molekyle, der syntetiseres i transkribtionfasen, kaldes mRNA (messenger RNA). Kort sagt er mRNA en kæde bestående af en enkelt streng af poly-nukleotider. Disse adskiller sig fra DNA ved at indeholde ribose i stedet for deoxyribose og basen uracil i stedet for thymin. De andre baser er ens i både mRNA og DNA. Transkription af DNA begynder med brydningen af hydrogenbindingerne, der holder de to DNA kæder sammen. Dette er ligesom med replikationen, dog brydes kæden på et meget mindre stykke. Denne brydning resulterer i, at baseparene nu er blottet og er frie til at kombinere sig med frie ribonukleotidtrifosfater: ATP, GTP, CTP og UTP. Adenin vil danne par med blottede thyminbaser på DNA, og på lignende måde vil frie ribonukleotidfosfater der henholdsvis indeholder guanin, cytosin eller uracil parre sig med de blottede DNAbaser cytosin, guanin og adenin. Urasil, som ikke findes i DNA, vil binde sig til basen Adenin. Den ene DNA kæde fungerer derved som en form for skabelon, der bestemmer nukleotidsekvensen på mRNA. Dette kan ses på Figur **??**.

Det er kun et af de to DNA strenge, der anvendes som skabelon, og derved dannes der kun 1 mRNA streng. Hvilken af de to DNA strenge, der skal anvendes som skabelon for

et bestemt gen bestemmes af promotoren, som sidder nær starten af genet. For et hvert given gen i kroppen er det kun én bestemt DNA streng, der benyttes som skabelon. RNA-polymerase begynder i promotorenden af et gen og flytter sig hen ad skabelon-strengen, og sætter ribonukleotider sammen på den voksende RNA-kæde, når der nåes en bestemt nukleotidsekvens, der indeholder en slutkode, frigives RNA strengen.

Translation

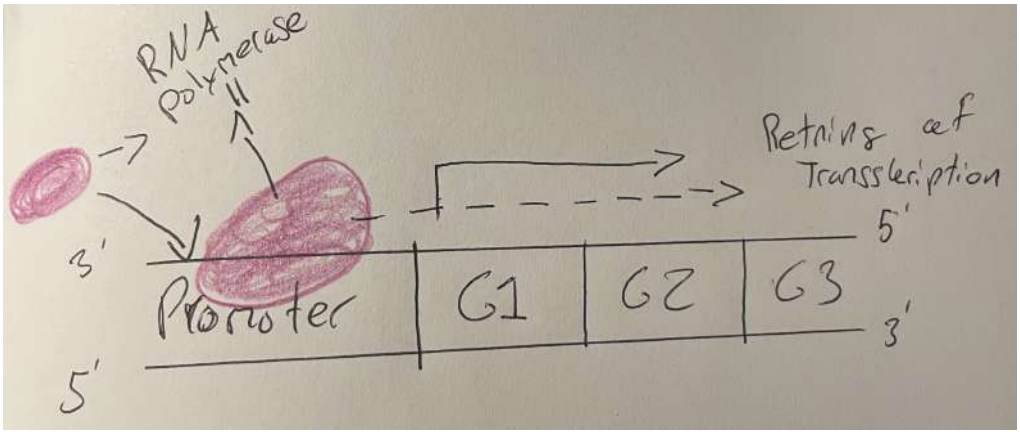
Når mRNA kæden er færdiggdannet bevæger den sig gennem nukliotidporerne i cellekernen og ud i cellens cytoplasma. I cytoplasmaet binder mRNA sig til et ribosom. Ribosomets funktion er at "oversætte" mRNAéts kode til et protein. Ribosomerne består selv af RNA (rRNA) og aflæser mRNA fra dets 5´er ende til dets 3´er ende. Hver sæt af tre baser kaldes et kodon. Kodoner koder for amino syrere. Det foregår sådan, at når et kodon bliver læst af ribosomet, så vil et stykke komplementært tRNA, som er bundet til en amino syre, blive rekruteret. Amino syren vil så blive tilført en voksende kæde af amino syrere. Ved slutningen af stregen vil der være et stop kodon, dvs et kodon som ikke koder for en amino syre, men som fortæller ribosomet at nu er proteinet færdigt. Ribosomet vil derefter give slip på både mRNA strengen og amino syre kæden, som nu vil folde sig og ende op med en bestemt struktur, klar til at blive anvendt til en bestemt funktion i kroppen.

4.4 Regulering af DNA udtrykning

Vi ved nu hvad DNA er og hvordan det bruges til at videregive information. Spørgsmålet bliver så, hvis alle celler indeholder alt DNA (for nemhedsskyld ignorerer vi gameter) for en given organisme, hvordan undgår vi så, at der ikke gror arme som ben og ben som arme. Eksemplet er lidt grelt, men det viser pointen. Vi bliver nød til at kontrollere hvilke dele af vores DNA, der bliver udtrykt hvor, og vi bliver nød til at styre hvornår. Disse spørgsmål bliver besvaret indenfor ting som genetik, embryologi og molekylær biologi. Her vil vi fokusere på hvordan en voksen organisme reagere på ændringer i omgivelserne. Dette kan være ting som hvilken mad der er til stede, eller hvilke hormoner der bliver udtrykt i kroppen. Det kan lyde fjernt fra arme der vokser der hvor ben burde, men de grundlæggende principper er de samme. Her vil vi gennemgå nogle simple eksempler på hvordan en celle kan regulere hvordan dens DNA bliver udtryks. De to eksempler er LAC-operonen for *E. coli* og transkriptionsfaktorer for eukaryoter.

LAC-operonen

LAC-operonen, eller laktose-operonen, er en operon i *E. coli*, som gør at *E coli.* kan nedbryde laktose. En operon er en samling af gener, ofte med relateret formål såsom at nedbryde laktose, under kontrol af en enkelt promoter. En promoter er en reguleringssekvens for en eller flere gener. Man kan tænke en promoter som et startfelt. Det er her RNA pol II binder for at udtrykke gener, men det er ikke en del af genet. Et gen er en DNA sekvens som koder for et protein. Se figur 4.4 for et eksempel.



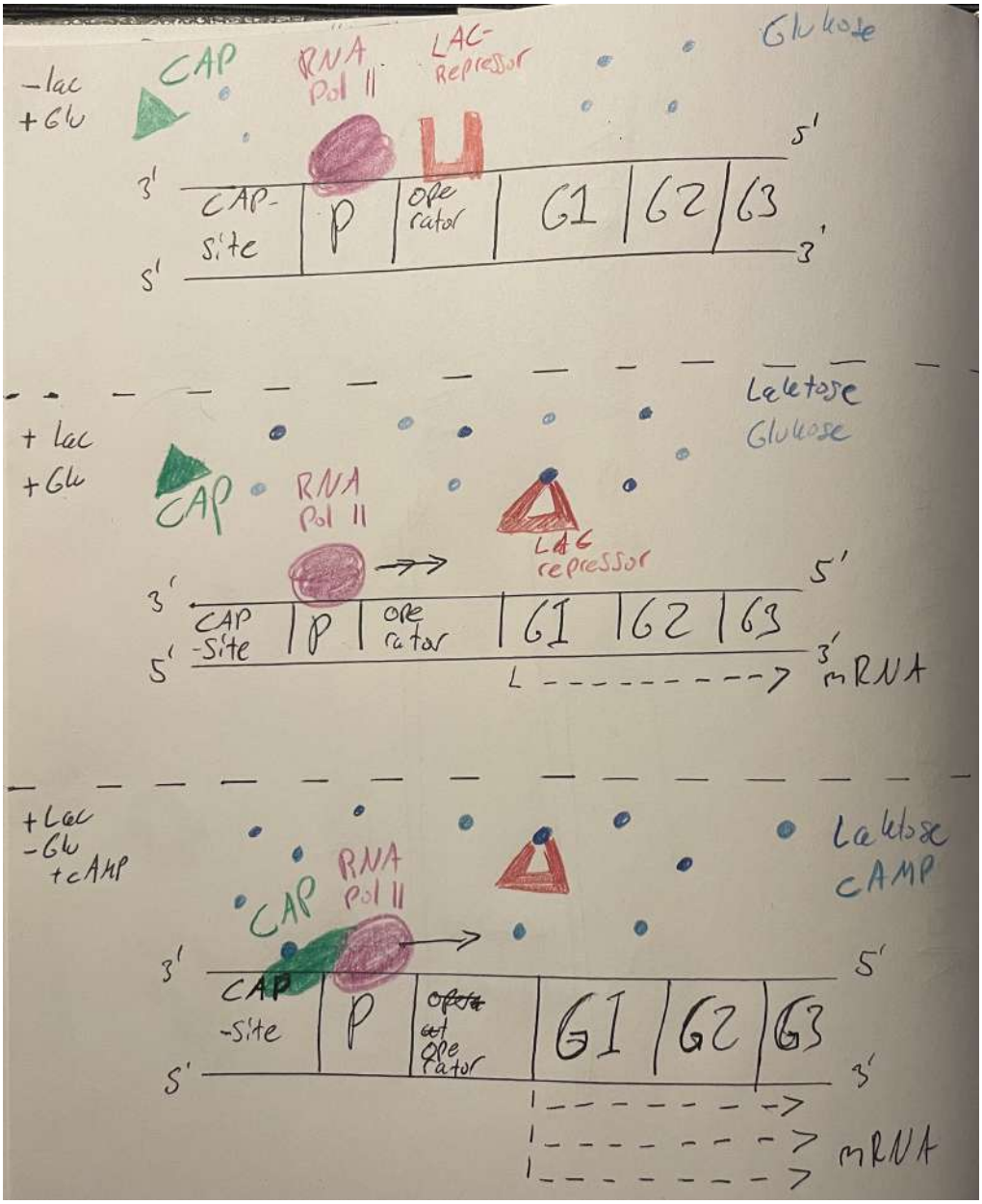
Figur 4.4: Skitsen viser en simpel operon, hvor tre gener som skal bruges til samme formål er kontrolleret af en fælles promoter.

Ligesom så mange andre, ønsker bakterier ikke at lave unødvendigt arbejde, og det at lave proteiner kræver mange kræfter og ressourcer. Derfor vil *E. coli* ikke udtrykke generne i LAC-operonen, medmindre det er nødvendigt, altså hvis der er laktose tilstede.

Måden det reguleres på, er ved en repressor og en aktivator. Vi ser på repressoren først. LAC-repressoren er et en tetramer, det vil sige 4 små proteiner som bliver samlet til et stort, og som sidder på det stykke af promoteren som kommer lige efter det stykke som RNA pol II skal binde til. Det betyder, at så længe repressoren sidder der, kan RNA pol 2 ikke bevæge sig langs DNAet og der kommer ingen transkription. Man kan tænke på repressoren som en stop klods på en model kørebane. LAC-repressoren har også et laktose-bindingsdomæne. Når laktose er til stede i mediet / miljøet, så vil det blive optaget over cellevæggen og binde til laktose-bindingsdomænet på LAC-repressoren. Når laktose binder sker der en konformationel ændring, det vil sige en ændring i protein strukturen. Denne ændring gør, at LAC-repressoren ikke længere kan binde DNA og giver derfor slip. Nu hvor LAC-repressoren ikke binder, kan RNA pol II bevæge sig langs DNAet og transkripere LAC-operonen, så *E. coli* kan nedbryde laktose.

Spørgsmålet er så hvad der sker, når der er både laktose og glukose tilstede. I det tilfælde vil *E. coli* først spise glukosen, så det giver ikke mening at udtrykke LAC-operonen. Vi ved, at når laktose er tilstede, så vil det binde til LAC-repressoren, som bliver frigjort af DNA'et så RNA pol II kan binde og bevæge sig langs DNAet. Men bare fordi, at RNA pol II kan bevæge sig langs DNAet, betyder det ikke, at det sker særlig effektivt. Det er her aktivatoren kommer ind i billedet. LAC-aktivatoren, som hedder catabolite activator protein (CAP) eller cAMP receptor protein (CRP), kan binde til CAP-sitet som kommer før den del af promoteren, hvor RNA pol 2 binder. Når CAP er bundet til CAP-sitet vil det interagere med RNA pol II og øge mængden af transkription.

Når der er glukose tilstede, er CAP ikke bundet til CAP-sitet, men flyder rundt inde i cellen. Det betyder, at hvis glukose og laktose er tilstede samtidig, så vil hverken LAC-repressoren eller CAP være bundet til DNAet. Med andre ord, der er transkription af LAC-operonen, men ikke særlig meget af det. Når glukose ikke er tilstede, vil noget af det første, der sker være, at cellen omdanner AMP til cAMP. AMP og ATP er en celles energikilder, ATP er et fuldt batteri, AMP er et tomt batteri. cAMP er et tomt batteri der er blevet lavet om til et cirkulært tomt batteri. Fordelen ved det er, at cAMP kan binde til en masse proteiner i en celle og regulere en masse forskellige ting. Blandt andet, så kan cAMP binde til CAP. Når cAMP binder til CAP, så sker der en konformationel ændring, som gør at CAP vil binde til CAP-sitet og øge transkriptionen af LAC-operonen. Figur 4.5 giver et hurtigt overblik over processen.



Figur 4.5: Skitsen viser LAC-operonen og dens regulative sekvenser.

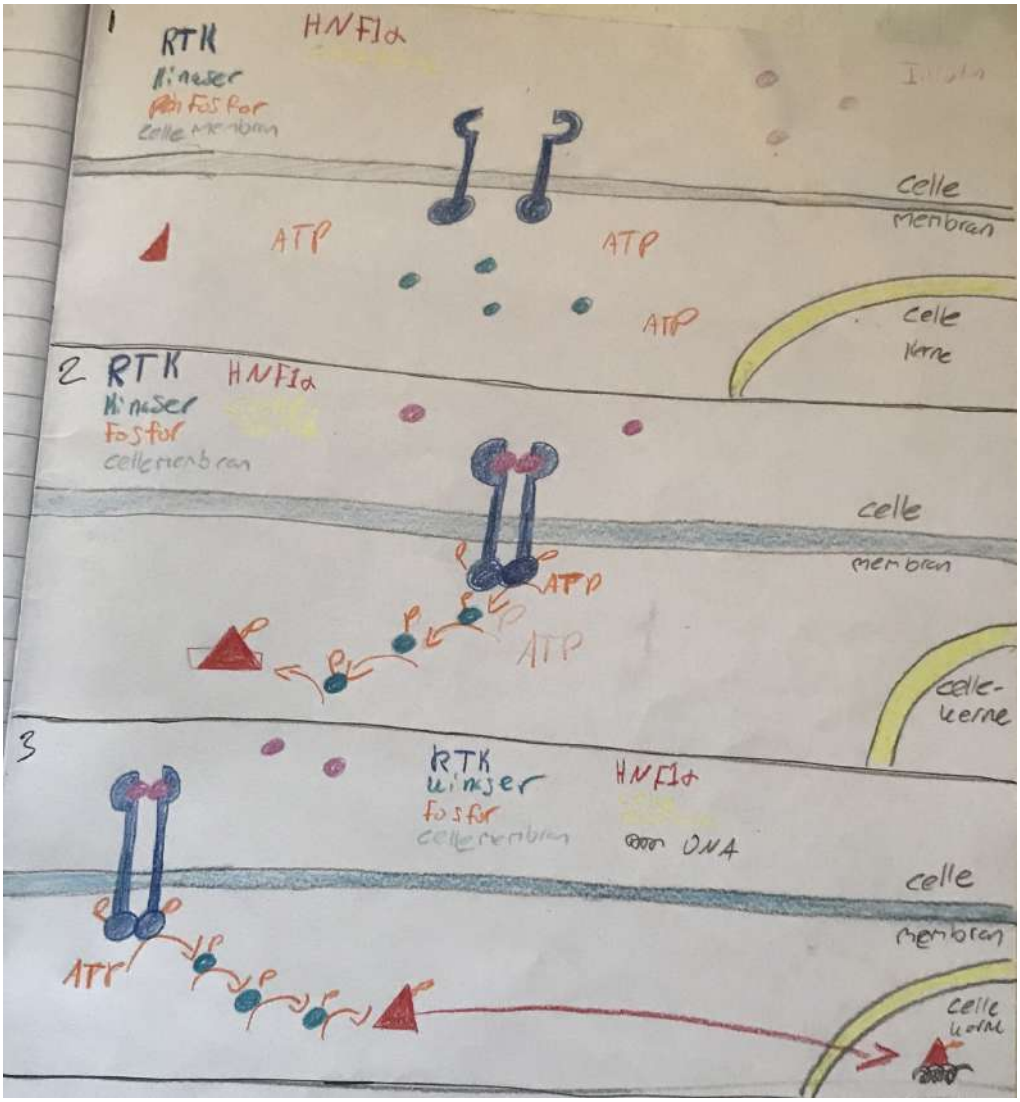
Spørgsmål Kan du tænke dig til hvorfor en bakterie hellere vil nedbryde glukose end laktose, når begge dele er til stede?

Transkriptionsfaktorer

Transkriptionsfaktorer er en gruppe proteiner, som har samme funktion som aktivatorene og repressorne i *E. coli*, de gør det bare på en anden måde. Ligesom i *E. coli* fungere transkriptionsfaktorer som tænd/sluk knapper, der styre om bestemte gener bliver udtrykt, men i modsætning til repressor /aktivator proteiner, så er transkriptionsfaktoren med til at binde RNA pol II til DNAet, i stedet for kun at interagere med allerede bundet RNA pol II. Årsagen er, at RNA pol II ikke er et enkelt protein, men nærmere et kompleks af mange små proteiner med hver deres rolle. Rollen med at genkende og binde til DNA er langt hen af vejen transkriptionsfaktorenes job.

Til at forklare processen, vil vi bruge MODY (Maturity Onset Diabetes of the Yo-

ung) som eksempel. MODY er fællesbetegnelse for en gruppe diabetes tilstande som har nogle andre karaktertræk end type 1 og 2. De træk er at det kommer ca. mellem 10-25 år, bugspytkirtlen fungere normalt og det løber i familier. Især det sidste er relevant for os, da det genetiske komponent ofte kommer fra en mutation i en transkriptionsfaktor. Disse mutationer gør, at en celle ikke kan reagere på insulin normalt. I en patient uden diabetes vil insulin binde til insulin receptorer. Disse receptorer kaldes receptor tyrosin kinaser (RTK), fordi de kan tilføje en fosforgruppe til amino syren tyrosin. De er bygget op med en recpetor del uden for cellen som kan binde signal molekyler, et transmembran domæne, et aktiveringsloop, og en kinase. Derudover er de aktive som homodimere, dvs en dimer bestående af to ens proteiner. Når insulin binder til en RTK sker der en konformationel ændring i receptoren. Denne ændring bringer de to aktiveringsloops tættere sammen og de aktivere derved hinanden ved at tilføje en fosfor gruppe. Den nye fosfor gruppe ændre strukturen lidt i kinase domænet, som derved bliver aktivt. Den aktive kinase starter en kaskade som til sidst aktivere transkriptionsfaktoren HNF1-α. HNF1-α er en transkriptionsfaktor, som kontrollere en masse forskellige gener i leveren og nyrerne. Disse gener er ansvarlige for behandlingen af glukose. Når HNF1-α bliver aktiveret, vil et NLS domæne blive frigjort. NLS står for nuclear localisation signal. Det er et domæne i et protein, som har en bestem amino syre sammensætning. Denne sammentsætning bliver genkendt af et protein kompleks, som transportere andre proteiner fra cytoplasmet og ind i cellekernen. Når HNF1-α først er i cellekernen vil det binde diverse promotere, rekruttere RNA pol II og lede til transkriptionen af de gener de kontrollere.



Figur 4.6: Den overstående skitse giver et overblik over hvordan insulin og transkriptionsfaktoren HNF1-α interagere. 1) insulin ikke til stede og som resultat er de to insulin receptorere adskilt. På samme vis af den interne kinase kaskade ikke aktiv, så HNF1-α er fanget i cytoplasmet og ingen af de gener den kontrollere bliver udtrykt. 2) Insuslin er til stede og har fået de to receptorere til at danne en homodimer. Som resultat af homodimeriseringen, er kinase aktiviteten blevet tændt og den interne kinase kaskade aktiveret. Dette er sket ved at tilføje fosfor grupper til dem alle. ATP donere de fosfor grupper som bruges. Til sidst får også HNF1-α en fosfor gruppe og bliver aktiv. 3) Den aktive HNF1-α transportes til cellekernen. Dette fører til transkription af en mængde forskellige gener (ikke vist)

I en patient med MODY, mere præcist MODY-3 som er den type som forsages af mutationer i HNF1-α, fungere den her mekanisme ikke. Det kan være, at det har været mutationer i NLS domænet. Det kan også være, at mutationerne er i POU-boxen, det domæne som binder til DNA. I alle tilfælde, resultere mutationerne i HNF1-α i, at en mængde gener ikke bliver udtryk og en patient vil præsentere med forhøjet blod sukker og manglende evne til at reagere på insulin.

Spørgsmål Kan du tænke dig til hvorfor en mutation i NLS domænen betyder at HNF1-α ikke kan udføre sin rolle?

4.5 Svar

DNA er det arvelige materiale: Fosfor og svovl er gode markøer, fordi DNA indeholder fosfor, men ikke svovl, mens proteiner indeholder svovl men ikke fosfor. Derfor kan man sige fosfor = DNA og svovl = protein.

Anvendelse af DNAs deling: Hvis flere celler deler sig end normalt, kan det være et tegn på tumor vækst og kraft.

Glukose vs Laktose: Glukose indeholder mere energi og det er den sukker vores metabolisme er mest gearet til at udnytte. Hvis man kigger på nedbrydningen af laktose i en bakterie, vil man ofte se, at den ikke bliver helt nedbrudt, men omdannet til glukose. Det betyder, at det er energibesparende at bruge glukose først, istedet for at omdanne laktosen til glukose for så at nedbryde glukosen.

NLS og HNF1-α: En mutation i NLS, betyder at HNF1-α ikke kan komme ind i cellekernen og føre til transkription.

ord.tex

Kapitel 5

Datalogi

5.1 Introduktion

Datalogi er et ord, som I måske ikke har hørt ofte. Dog har datalogien og dets væsen en større indflydelse på vores samfund end nogensinde før. Datalogi er egentlig blot den danske version af ordet computer science. Datalogi beskæftiger sig med den teoretiske forståelse af computeren fra hardware til software, og alt derimellem. Det er den digitale logik, 1'ere og 0'ere og hvordan de kan opereres på med logiske operationer, men også hvordan man sorterer tal i en liste. I dette afsnit kommer vi til at lære noget indledende programmering i Python, som er det praktiske del af datalogi.

Hvem er vi og hvad betyder datalogi for os?

Jin

Jeg har siden 8.-9. klasse programmeret Discord bots som en hobby. Jeg blev introduceret til programmering da jeg så hvordan man kunne bygge disse "chatbots", som kunne svare tilbage da brugeren skrev en besked. Jeg syntes, at det var sjovt at bygge disse chatbots, og derfra kunne jeg lære en del programmering selv. Jeg arbejder i dag som programmør ved siden af mit studie, og jeg har kunne bruge ret meget af mine erfaring fra at programmere Discord bots dengang i folkeskolen, til at nu lave noget mere fornuftigt. Programmering for mig handler om at undersøge problemer og derefter at kunne løse dem, i sær har jeg lært meget af mine problemløsnings kompetencer fra programmering, og det er noget, som jeg vil råde enhver at lære.

Louie

Siden helt lille har jeg været fascineret af matematik og dens underliggende betydning for vores verden. Datalogi var noget jeg mødte for første gang i gymnasiet, hvor valgte at tage programmeringslinjen. Programmering var et fag jeg meget hurtigt blev glad for, fordi jeg fandt ud af at det ikke bare et værktøj, som kunne benyttes til at skabe hjemmesider, men viden og applikation, som kunne bruges til løse udfordrende problemer igennem diverse algoritmer. I dag studerer jeg fysik, og bruger ofte datalogi til studiet, blandt andet til simulation af planeters baner samt avancerede former for databehandling.

Mit gode råd for en person som gerne vil lære at programmere er at skrive men masse programmer, enten som små projekter eller løsninger til programmeringsproblemer.

Hvad er datalogi og programmering

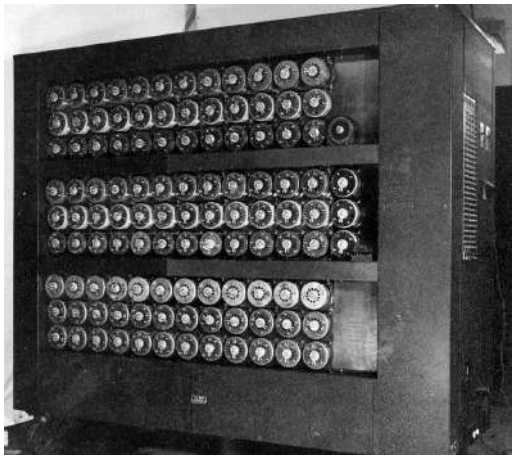
Dette afsnit er en kort introduktion til hvad programmering er, og kan anbefales for alle at læse, før de kaster sig ud i det. For at forstå hvad programmering er, skal vi først forstå hvordan en computer fungerer.

Historien af computeren

Digitale computere som vi bruger i dag har sin oprindelse tilbage fra anden verdenskrig, hvor de allierede var desperate for at kunne knække det såkaldte "Enigma-koden", som tyskerene brugt til at kommunikere med hinanden. Mange har måske hørt om Alan Turing, som hyldes for at være grundlæggeren for moderne datalogiske principper.

Figur 5.1: Alan Turing da han var 16

Turing var netop den der forsøgte at designe således en maskine under anden verdenskrig, og det lykkedes for ham at opfinde en maskine der kunne knække tyskernes Enigma, hvilket havde stor betydning for at de allierede kunne vinde krigen.



Figur 5.2: Maskinen "Bombe", som Turing designede for at knække Enigma

Bombe var ikke en computer i nutidens forstand da den havde meget begrænset mulighed for at beregne alt andet end at knække Enigma-koder, mere specifikt kunne den ikke udvides til en "Turing-maskine". En Turing-maskine er en abstrakt maskine der teoretisk ville kunne udføre enhver slags beregning som man kan finde på. Turing-maskiner eksisterer ikke i virkeligheden, og er blot en koncept, men de har inspireret udviklingen af moderne computere. Nutidens computere kan udvides til en Turing-machine i den forstand at hvis de havde adgang til uendelige meget hukommelse/lagerplads, så ville de være ækvivalent med en Turing-maskine. Det samme kan man ikke sige om Bombe, da den i meget begrænset omfang var kun målrettet til at kunne knække Enigma-koder. I dag med den processorkraft som vi har i 2022, kan vi faktisk simulere Bombe på vores egne bærbare computere!

Hvad er en computer?

Computere virker ved, at vi mennesker fortæller den hvad den skal gøre, altså hvilke slags beregninger den skal køre for at nå frem til et resultat. Det er at *manipulere/udføre beregninger på* ***information***, som reelt set udgør formålet ved en computer. Det er også derfor at ordet IT benyttes, da det er en forkortelse for informations-teknologi. Information kan i nemmeste forstand, tænkes som et stykke viden som har en betydning for hvordan man *handler* i verden. Fx kunne jeg sige til dig "Det regner virkeligt meget udenfor". Det kan være nyttigt for dig at vide det, hvis du skal planlægge hvorvidt du bør tage regnjakken på. Din handling er så det at tage regnjakken på, som afhænger af informationen om hvorvidt det regner. Hvis en computer skulle kodes for at kunne fortælle dig hvad du skal gøre når det regner, så kunne det fx være:

```
regn = True
if regn:
    print("Tag regnjakken på")
else:
    print("Det regner ikke, du behøver ikke en regnjakke")
```

Dette er en meget simpel *algoritme*, og de fleste kan nok regne ud, at de bør tage regnjakken på hvis det regner. Men når informationen man er givet bliver mere og mere kompleks bliver man nødt til at regne mere for at tage beslutninger, og det er her træder computere ind i billedet. I den virkelige verden har man brug for computere og algoritmer til at fortælle hvad der præcist skal gøres under hvilke omstændigheder. Det kan fx. være at man i en fabrik har en computer, der styrer en robot, som hælder noget kemikalie i et glas for at frembringe en kemisk reaktion. Her kan det være meget vigtigt at man holder øje med, hvor meget kemikalie der er i glasset, hvad man har hældt i glasset, samt hvor lang tid er der gået. Disse størrelser udgør den konkrete information som en computer kan regne på for at finde ud af helt præcist hvor meget den skal hælde i for at glasset ikke eksploderer.

Hvad er programmering?

Programmering er så måden man udtrykker til computeren hvordan den skal behandle den information som den modtager. I programmering skriver man i et *programmeringssprog*, som er et sprog som computeren kan forstå for at vide helt præcist hvad den skal gøre. For at kunne programmere computeren til at gøre noget, skal man kunne snakke dens sprog. Vi vil her komme til at beskæftige os med programmeringssproget Python, som er et meget populært og nemt sprog at lære, og er nok er det allermest brugte programmeringssprog i hele verden!

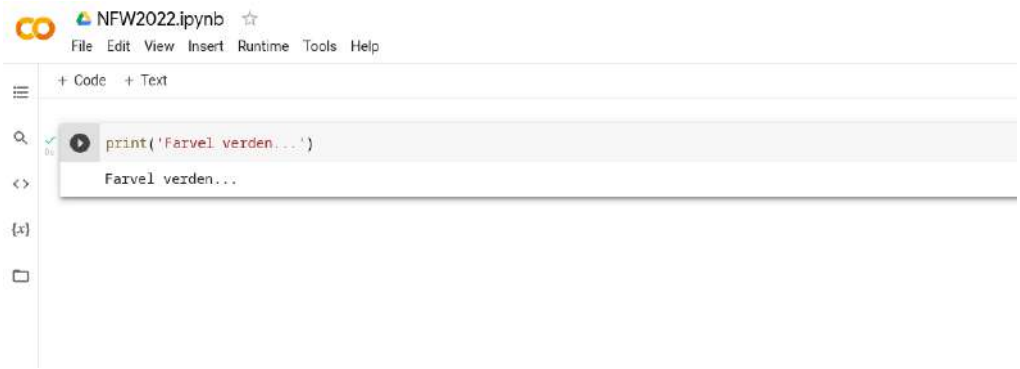
Når man programmerer, oversætter computeren fra det programmeringssprog man benytter videre til maskinkode, som er de specifikke *instruktioner* computeren tænker at den skal udføre. Det er ligesom, at man fx. kan snakke med en tysker hvis man snakker engelsk, men man tænker selv i dansk, og tyskeren tænker i tysk, dog har I et fælles sprog, engelsk, som I begge kan forstå. Med computeren bliver det klart, at det fælles sprog som I begge kan tale er Python. Men mere korrekt skal man forstå, at dette er mere eller mindre en en-vejs kommunikation, computeren gør det du fortæller den at den skal gøre, og den rapporterer så tilbage resultatet af arbejdet. I regnjakke-eksempelet ovenfor bad vi computeren om at skrive: "Tag regnjakken på" hvis regn variablen var sand, ellers ville den skrive "Det regner ikke, du behøver ikke en regnjakke".

Tabel 5.1: Relevante termer og deres betydning

Ord	Definition
Information	Noget viden der gør at du handler på en bestemt måde
Computer	En maskine der manipulerer på den information som vi mennesker giver den
Turing-maskine	En hypotetisk maskine der er i stand til at beregne alt hvad vi kan finde på, alle moderne computere kan udvides til Turing-maskiner
Programmering	Disciplinen i at kommunikere med computeren - at fortælle den hvad den skal gøre
Programmeringssprog/kode	Det man snakker for at kommunikere med computeren
Maskinkode	Instruktioner/ord som computeren forstår
Python	Et programmeringssprog udviklet i 1991 af Guido van Rossum
Python-interpreter	Et program som oversætter Python kode til maskinkode
Google-Colab	Online værktøj udviklet af Google for at køre kode på Google's computere

Konsollen og print funktionen

Du er nu klar til at skrive dit første stykke Python-kode. Vi benytter Google-Colab's Jupyter Notebooks for at gøre dette. Det er en online Python-editor hvor man kan skrive Python kode, og derefter eksekvere koden.



Figur 5.3: Google-Colab, hvor vi printer en streng

I billedet oven skriver vi koden `print('Farvel verden...')`, hvorefter resultatet, som er teksten, "Farvel verden...", ville blive "printet" til konsollen linjen under. Konsollen er der hvor Python udskriver resultatet for dens beregninger. For at printe noget til konsollen skal man bruge `print` funktionen, som er en indbygget i Python. Teksten, som man printer, kaldes for en `string`, som bare er Pythons ord for et stykke tekst. Teksten i en `string` skal man altid skrive i mellem apostrofer, eller gåseøjne, `''`, `""`. Man kan bruge `print` på følgende måder:

```
# Prøv at skrive disse i Google-Colab
print('Normal print med \', man skal skrive \' for at Python
    ↪  ikke tror at det er slutningen på ens streng ')
print("Normal print med \", man kan også bruge gåseøjne frem
    ↪  for apostrof")
```

```
# Hvis man skriver , kan man print noget ekstra, fx et tal
    ↪ lige efter ens string
print('Min alder er', 19)

# Man kan også bruge +, her skal man dog først konvertere et
    ↪  tal til en string
print('Jeg er '+str(19)+' år gammel')
```

Oven har vi nogle gange skrevet #, hvilket Python tolker som en *kommentar*. Kommentarer har ingen påvirkning på ens kode, og er der kun for at hjælpe programmører med at huske hvad ens kode gør. Man har også mulighed for at få Python til at tage imod bruger-input, dvs. at man kan få Python til at køre noget kode baseret på data man får fra brugeren. Forsøg fx at køre følgende

```
navn = input('Hvad er dit navn? ')
print('Hej '+navn+', jeg glad for at møde dig!')

# Man kan også skrive, men så laver Python selv mellemrum
print('Hej', navn, ', jeg glad for at møde dig!')
```

Man får så lov til at skrive ens navn i konsollen, derefter vil Python printe ens navn og at den er glad for at møde dig.

Opgaver til konsollen og print funktionen

●● **Opgave 5.1.1:**
Hvorfor virker følgende program ikke, kan du fikse det? (Se evt. næste kapitel)

```
tal = input('Giv mig et tal ')
print('Dit tal +5 er', tal+5)
```

5.2 Typer og variabler

Datatyper

Hvad er en datatype? En datatype er en klassificering af hvilken type data man har at gøre med. Der er 4 basale datatyper i Python, de er:

1. `int`, som repræsenterer heltal, fx, `-1, 2, 44, 42, 69`
2. `float`, som er decimaltal, fx, `-1.12, 3.1415, 1.4423, 2/3`
3. `boolean`, som kan antage to sandhedsværdier, `True` for sandt, og `False` for falsk
4. `str`, som er et stykke tekst, fx, `"Se mor! Jeg koder i Python!", "Jeg er 14 ↪ år gammel"`
5. `NoneType`, ingenting, hvis variablen er `None`.

Disse datatyper er dem som vi vil primært beskæftige os med. Typisk har man brug for at konvertere fra en datatype til en anden. Fx kan Python ikke tolke `print('9'+10)` ↪ da man forsøger at plusse en `str` med en `int`, skal den printe `'910'` eller `19`? Det ved den simpelthen ikke. Derfor opstiller man i programmeringssprog klare konventioner for hvordan operationer skal udføres. For at kunne gøre ovenstående, skal man derfor enten konvertere tallet 9 fra `str` til `int` eller omvendt for 10. Det kan man gøre ved at skrive, `print(int('9')+10)` eller `print('9'+str(10))`. Man kan konvertere alt fra en bestemt datatype til en anden, hvis man bare skriver `t(_)` hvor `t` kan være alt mellem `int, str, bool, float` og `_` er det tal/streng man vil gerne konvertere.

Variabler

Variabler bruger man til at gemme en værdi, som kan antage de datatyper beskrevet oven:

```
fem = 5
to = 2

syv = fem + to
print(syv)
>>> 7
```

For at danne en variable, skriver man først navnet på ens variabel, som i eksemplet foroven var `fem` eller `syv`, variabel navnene kunne ligeså godt være kaldt hhv. `tallet_fem` eller `antal_dage_i_ugen`. Derefter skriver man et lighedstegn `=` følgende med værdien man gerne vil definere ens variabel med. Her kan man se flere eksempler

```
mit_yndlingstal = 420
tallet_pi = 3.14159265359
det_er_sandt = True
vores_navne = "Jin og Louie"

# Det her er en kommentar, som ikke ville blive kørt i koden
# Man kan lave matematik med variabler
nyt_tal = mit_yndlingstal + tallet_pi
print(nyt_tal)
>>> 423.14159265359
```

Nogle kvikke har måske bemærket, at vi ikke skriver med mellemrum " ", når vi definerer variabler. Det er noget, som Python klager over, hvis man gør (bare prøv at gøre det hvis ikke I stoler på os!). Der findes visse begrænsninger for variablernes navne, alle variabler skal opfylde følgende

1. Den kan enten starte med underscore, `_` eller et bogstav. Man kan ikke starte en variabel med et tal!

2. Variabler må kun indeholde bogstaver, tal eller underscore, `_`.

3. Variabler er "case-sensitive", dvs. `TALLET_fem` og `tallet_fem` er to forskellige variabler.

Man kan fx se et eksempel her

```
# Lovlige navne på variabler:
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"

# Ulovlige variable navne, ALDRIG skriv disse:
2myvar = "John"
my-var = "John"
my var = "John"
```

Matematiske operationer

Variabler er en meget nyttige at bruge når man skal lave regnestykker, som kræver at man kan referer til noget data som man har gemt før, fx i matematik kan man lave andengradsligninger, hvor der indgår variablerne a, b, c og x i ligningen $ax^2 + bx + c = 0$. I Python kan man udregne matematiske ligninger og funktioner ved at definere variabler, og så derefter lave matematiske operationer på dem:

```
a = 5
b = 3
c = -9

# Vi udregner andengradsligningen her (det hedder rent
    ↪ faktisk et andengradspolynomium...) for x lig 4

x = 4
resultat = a*x**2 + b*x + c

print(resultat)
>>> 83
```

På linje 8 skrev vi koden `a*x**2 + b*x + c`, som repræsenterer den matematiske formel $ax^2 + bx + c = 5 \cdot 4^2 + 3 \cdot 4 - 9 = 83$. Der er visse operationer som man kan lave, her kan I se en liste af dem

```
# Tal
3  # => 3

# Matematik som man forventer det
1 + 1   # => 2
8 - 1   # => 7
10 * 2  # => 20
35 / 5  # => 7.0

# Når man dividerer 2 tal får man altid en float
10 / 3  # => 3.3333333333333335

# Modulo operation, division med rest
7 % 3   # => 1, fordi 2 gange 3 = 6, som er 1 væk fra 7

# Potensløftning (x**y, x opløftede i det y'te potens)
2**3  # => 2*2*2=8

# Parentes for hvilken operationer man regner først
1 + 3 * 2    # => 7
(1 + 3) * 2  # => 8
```

Man kan lave ret avancerede matematiske ligninger, fx idealgasligningen (noget I kommer til at lære i gymnasiet), $PV = nRT$. Her kan man isolere $n = \frac{PV}{RT}$ og udregne n givet at man har defineret variablerne P, V, R, T

```
P = 197 # Atmosfærisk tryk
V = 22.4 # Liter (vi definerer en float her)
T = 28 + 273 # Kelvin
R = 8.21 * 10**(-2) # konstanten R, videnskabelige notation

n = (P*V)/(R*T)
print('n har vi udregnet til at være', n, 'mol')
```

```
# Hvis man ikke havde variabler, så skulle man skrive noget
    ↪ rod i denne stil
n = (197*22.4)/((8.21 * 10**(-2))*(28 + 273))
# Som er uoverskueligt da man ikke ved hvad tallene betyder
```

5.3 Sammenligninger og udtryk

https://docs.python.org/3/reference/expressions.html#comparisons Noget af det mest fundamentale i programmering er at man skal kunne sammenligne to værdier med hinanden for at afgøre hvorvidt noget kode skal køre eller ej. I matematik, kan man fx opstille *udtryk* såsom:

$$\text{sand} : 5 < 10$$
$$\text{falsk} : 1 = 2$$
$$\text{sand} : 6 \cdot 5^2 + 4 \cdot 5 + 9 > 0$$

Hver af disse linjer har en *sandhedsværdi* som er hvorvidt udsagnet er sandt eller falsk. Det er sandt at 5 er mindre end 10, derimod er det falsk at 1 er lig 2. Bemærk, at ethvert af disse udtryk, består netop af 3 dele, 2 værdier og et tegn der sammenligner disse to værdier. I det første har vi værdierne "5" og "10", som bliver placeret på hver sin side omkring mindre-end tegnet "<". I det allersidste udtryk foroven, har vi andengradspolynomiet $6 \cdot 5^2 + 4 \cdot 5 + 9$ som antager værdien 179 hvis man udregner den, og som er selvfølgelig større end 0.

I Python findes der logiske *udtryk*, som har en af to værdier. `True` for sand, eller `False` for falsk. Fx kunne nogle af disse udtryk se ud på følgende måder

```
print(1 == 2)
>>> False

print(5*10-10 < 40)
>>> False

print('Ingen stavefejl her' != 'Ingen stavefejl hr')
>>> True

tid = 800
print(tid < 900)
>>> True
```

Vi kalder de tegn man bruger til at sammenligne to værdier med hinanden for sammenligningsoperatorer, eller comparison operators på engelsk. Her er en liste af alle de vigtige sammenligningsoperatorer.

Tabel 5.2: Tabel af sammenligningsoperatorer

Operator	Tilsvarende i matematik	Eksempel
`==`	$=$	`5 == 4` er `False`
`!=`	$\neq$	`5 != 4` er `True`
`>=`	$\geq$	`5 >= 4` er `True`
`<=`	$\leq$	`5 <= 4` er `False`
`>`	$>$	`5 > 4` er `True`
`<`	$<$	`5 < 4` er `False`

Som man så i eksemplet foroven, så kan man fx. godt sammenligne `string`s med hinanden, hvilket kunne være meget nyttigt hvis man fx skulle teste om hvis en bruger

har tastet noget korrekt. Desuden sammenligner man for det meste af tiden, en variabel med en anden værdi, som også kunne være en variabel. Dette gør, at vi kan opstille betingelser for hvornår noget kode skal køre, alt afhængigt af hvilken værdi variablen har (se næste kapitel).

Vi skal nu til at snakke om boolske operationer, som er operationer der sammenligner og negerer *udtryk*, hvor vi førhen havde sammenligningsoperationer, som sammenlignede *værdier*, og gav et *udtryk*, bruger vi boolske operationer på selve udtrykkene. De tre mest basale boolske operationer er, `not`, `and` og `or`. Operationen `not` negerer den boolske værdi, hvilket ville sige at et `False` bliver til et `True`, og omvendt bliver `True` til `False`. Operationen `or`, sammenligner to udtryk, og hvis blot en af dem er `True`, så returnere den `True`. Kun hvis begge er falske returneres `False`. Sidst, men ikke mindst, så er der `and`. `and` sammenligner ligesom `or`, to værdier. `and` returnerer kun `True`, hvis begge værdierne af sande, ellers returneres `False`. For at give et eksempel så betragte følgende. Lad A og B være vilkårlige udtryk, de kunne fx være `5*5 == 25` eller `tid < 1000`, hvor `tid` er en variabel. Disse udtryk har så en sandhedsværdi, og vi kan opstille en tabel hvor vi bruger de boolske operationer, `and` og `or` på disse udtryk:

Tabel 5.3: Tabel for boolske operationer

A	B	`or`	`and`
`True`	`True`	`True`	`True`
`False`	`True`	`True`	`False`
`True`	`False`	`True`	`False`
`False`	`False`	`False`	`False`

Vi kan nu fx skrive mere komplicerede udtryk såsom disse:

```
tid = 900 # kl 9.
penge = 1000 # kr.

print(tid < 1000 and penge == 800)
>>> False

tid_er_lig_penge = tid == penge # man kan få variabler til
    ↪ at gemme resultatet af et udtryk
print(tid + 100 == penge or tid_er_lig_penge)
>>> True

print(not (tid + 100 == penge))
>>> False

print(tid + 100 != penge) # det samme som oven
>>> False

print(not (tid == penge and penge-100 < 100))
>>> True
```

Opgaver til datatyper og udtryk

- **Opgave 5.3.1:**
  Udregn følgende i Python og derefter print resultatet.

  1. $939577132 \cdot (2124 - 65)$
  2. $129^3 \cdot 2124 - \frac{4}{3}$
  3. 420^{69}

4. $\frac{1}{\frac{1}{\frac{1}{3}}}$

5. 5^{5^5}

●● **Opgave 5.3.2:**
Tænk over resultatet (og om det kan evalueres og evaluer bagefter i python.

1. `(not False and True)or False`

2. `not not not not False`

●●● **Opgave 5.3.3:**
For vilkårlige udtryk `A` og B, er det korrekt at

1. `not (A and B)== not A or not B`

2. `not (A or B)== not A and not B`

Altid evaluerer til `True`?
Svar: ja, denne lov kaldes for De-Morgans lov, overvej hvorfor

5.4 Control sequences, if, else, for, while

Vi skal nu kigge på control sequences, som er måden man styrer hvilken kode skal køre under hvilken betingelser. Det er vigtigt at man mestrer brugen af disse når man skriver sine programmer.

if-else udtryk

Vi starter med at forstå hvad et if-else udtryk er, som er et stykke kode struktureret på følgende form:

```
mit_yndlingstal = 42
if mit_yndlingstal > 50:
    print("Mit yndlingstal er større end 50")
elif 40 < mit_yndlingstal < 50:
    print("Mit yndlingstal er mellem 40 og 50")
else:
    print("Mit yndlingstal må være mindre eller lig 40")
```

Ordet "if" er "hvis" på Dansk, og "`elif`" er blot en forkortelse af "else-if", hvilket betyder "ellers-hvis" på Dansk. Til sidst har vi "else" som betyder ellers. Det er ikke særligt svært at gennemskue hvad der sker i koden. Der bliver først defineret en variabel, som er `mit_yndlingstal` som er sat til `42`. I det første `if` skriver vi en *betingelse*, som reelt kan være hvilket som helst udtryk. Betingelsen i dette tilfælde er `mit_yndlingstal > 50`, og hvis den er opfyldt, dvs. hvis variablen som hedder `mit_yndlingstal` er rent faktisk større end 50, så ville der blive printet strengen, "Mit yndlingstal er større end 50", til konsollen. Den første if-udtryk kan læses meget direkte som: "Hvis mit yndlingstal er større end halvtreds, så kør koden neden hvor vi printer ... til konsollen".

Bemærk her at der er 4 mellemrum (en tab) under linjen efter vores `if`, `elif`, `else`, dette er nødvendigt for Python at vide hvilket stykke kode den skal køre, hvis betingelsen er opfyldt. Fx er disse ikke lovlige if-else udtryk:

```
mit_yndlingstal = 42

# Python ville sige at der er en fejl, prøv at køre hvis du
    ↪ ikke tror på os!
if mit_yndlingstal > 50:
print("Mit yndlingstal er større end 50")
elif 40 < mit_yndlingstal < 50:
print("Mit yndlingstal er mellem 40 og 50")
else:
print("Mit yndlingstal må være mindre eller lig 40")
```

I dette tilfælde er `mit_yndlingstal` dog ikke 50, den er 42, så her kommer `elif` ind i billedet. Et "else-if" bliver kørt hvis ens første "if" ikke var opfyldt. Så vil Python kigge *sekventielt* på det næste stykke betingelse den skal evaluere for at køre noget, hvilket i dette tilfælde er det der står i `elif` udtrykket, som er blot et `if`, men som ikke er det første. Man kan have vilkårlige mange `elif` udtryk, og det gør at man kan kontrollere meget præcist hvad skal køres under hvilke omstændigheder. Til allersidst har vi `else`, som er det der køres hvis alt andet fejler, dvs. hvis alle ens `if`, `elif` udtryk ikke blev opfyldt, her behøver man ikke at skrive en betingelse.

Opgaver til if-else udtryk

- **Opgave 5.4.1:**
Hvilket af følgende er korrekt brug af if-else?

```
# 1
else:
    print("Hej med dig")

# 2 Overvej hvad der ville blive printet i den følgende
mit_yndlingsfarve = "rød"
if mit_yndlingsfarve != "rød":
    print("Mit yndlingsfarve var ikke rød!")

# 3
louies_tal = 420
jins_tal = 40
if louies_tal < 500:
    if jins_tal < 500:
        print("Både Louie og Jin har et tal under 500")
else:
    print("Louies tal er ihvertfald større end 500, vi ved
       ↪ ikke om Jin")

# 4 Overvej om hvis if-udtrykket (kig bort fra else) i 3 er
    ↪ det samme som
if louies_tal+jins_tal < 1000:
    print("Både Louie og Jin har et tal under 500")

# og/eller
if louies_tal < 500 and jins_tal < 5000:
    print("Både Louie og Jin har et tal under 500")
```

5.5 for og while loops

Vi skal nu se på for og while loops (løkker på Dansk), som er måden man kan få Python til at køre det samme kode flere gange. En for-loop antager følgende form:

```
# 1 Printer alle tal fra 0 til 9, 10 er ikke inklusive
for tal in range(0, 10):
    print("Hej tal:", tal)

# 2 Man bruger for loops til at iterere over lister, som i l
   ↪ ærer i næste kapitel
# Koden neden er samme som 1
nul_til_ni = [1, 2, 3, 4, 5, 6, 7, 8, 9]
for tal in nul_til_ni:
    print("Hej tal:", tal)

# 3 Man kan have lister af alt muligt, her bruger vi strings
louie_og_jin = ["Louie", "Jin"]
for unfer in louie_og_jin:
    print("Hej", unfer)
```

Hvis man får Python til at køre koden foroven, så vil den printe strengen: "Hej tal: _", hvor _ løber fra 0 til 9. Når man skriver en for-loop, *itererer* man over *elementer*, som fx kunne være tal eller eller elementer i en liste. Hvis man vil gerne tælle fra et bestemt heltal a til et tal b, kan man bruge Python's indbygget funktion "`range`", til at skrive `range(a, b)` som vil *genererer* alle tal fra a til b hvor b ikke er inklusive. Det er helt afgørende når man skriver for-loops, at man bruger operatoren "`in`", som udtrykker til Python at man vil gerne itererer over elementer der *ligger i* en liste eller `range`. I det første eksempel oven, kan vi læse for-loopen som "for alle tal i mellem 0 til 10, print "Hej tal: _" til konsollen". Vi bevæger os videre til while-loops, som er en anden måde man kan skrive løkker på

```
tal = 0
while tal < 10:
    print("Hej tal:", tal)
    tal += 1
```

Med while-loops er der tale om en betingelse, som skal være opfyldt for at while-loop'en må køre. I eksemplet foroven, har vi en while-loop som betinger sig på at `tal` skal være mindre end 10, i hvilket tilfælde ville den så printe tallet fuldstændigt tilsvarende til det for-loop vi så før. I selve while-loop'en har vi netop koden `tal += 1` som svarer til `tal = tal + 1`, som I måske kan regne ud hvad det betyder.

En for loop, kører på en anden slags betingelse end en while-loop, den væsentligste forskel er at en for-loop iterer over elementer, så længe at der er elementer til rådighed, kører for-loop'en. En while-loop derimod kører på at while-udtrykket er opfyldt, som skal være opfyldt så længe at man har lyst til at køre koden.

Opgaver til for og while loops

●●● **Opgave 5.5.1:**
Overvej hvad der ville ske med koden nedenunder, prøv evt. at skrive dem i Python!

```
# 1
for tal in range(7, 5):
   print(tal)

# 2
tal = 0
```

```
for tal < 10:
    print(tal)

# 3 Tænk grundigt over følgende!
tal = 8
while tal < 10:
    print(tal)

# 4
unf = ["Louie", "Jin", "Rasmus"]
for unfer in unf:
    print("Vi har fat i", unfer)
    if unfer == "Louie" or unfer == "Jin":
        print("Han er nemlig en datalog!")
        if unfer == "Louie":
            print("Og han kan godt lide kompetitiv
               ↪ programmering")
        elif unfer == "Jin":
            print("Og han kan godt lide HTML programmering")
    else:
        print("Hmm, gad vide om hvis vi har fat i den
           ↪ rigtige person? Mon ikke det er matematikeren
           ↪ Rasmus?")
```

5.6 Datastrukturer: lister og dictionaries

I denne sektion skal vi kigge på nogle simple datastrukturer: lister og dictionaries.

Lister

Så hvad er en liste? En liste er blot en samling af elementer, den kan være vilkårligt lang og man kan tilføje og fjerne ting fra listen. Man bruger lister til at organisere ens data, fx hvis man har data der skal læses i en bestemt rækkefølge for at det giver mening. Her kan vi se hvordan vi definerer en liste med nogle elementer.

```
primtallene = [2, 3, 5, 7, 11, 13] # En liste med nogle
   ↪ primtal
ugens_dage = ["Monday", "Tuesday", "Wednesday", "Thursday",
   ↪ "Friday", "Saturday", "Sunday"] # En liste med ugens
   ↪ dage
forskellige = ["Monday", 1, "Tuesday", 2] # Man kan have
   ↪ forskellige datatyper i en liste
# Man behøver ikke at have noget i listen når man definerer
   ↪ den
# Det kan være at man senere vil gerne tilføje til den.
tom_liste = []
```

Så hvad kan man gøre med lister? Man kan indicere, indsætte, fjerne og meget andet. Nedenfor kan nogle eksempler ses. Her er det vigtigt, at nævne at lister er indicerede fra 0 af, hvilket betyder at man tilgår det første element ved at referere til den 0 position i listen.

```
# Indicere (tilgå et element)
print(primtallene[0])
>>> 2
```

```
# Udskifte (Udskifte et element)
ugens_dage[1] = "Tirsdag" # Husk vi starter ved element 0, s
    ↪ å er Tuesdag nr. 1 i listen
print(ugens_dage[1])
>>> Tirsdag

# Appende (Indsætte et element i slutningen af listen)
tom_liste.append("Hej")
tom_liste.append("med")
tom_liste.append("dig")
print(tom_liste)
>>> ['Hej', 'med', 'dig']

# Sæt to lister sammen
hej1 = ["Hej", "med", "dig"]
hej2 = [",", "hvordan", "går", "det?"]
hej3 = hej1+hej2
print(hej1+hej2)
>>> ['Hej', 'med', 'dig', ',', 'hvordan', 'går', 'det?']

# Slette
søndag = ugens_dage.pop() # Fjerner allersidste element og
    ↪ returnere den, som er "Sunday"
print(søndag)
>>> Sunday
mandag = ugens_dage.pop(0) # Fjerner det første element
print(ugens_dage)
>>> ['Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday
    ↪ ']

# Dette må du ikke gøre!
print(primtallene[10]) # Dette element eksisterer ikke (out
    ↪ of bounds)

primtallene[10] = 15 # Du kan igen ikke tilgå a[10]
```

Vi kan gøre mange ting med lister, hvis vi fx vil gerne udregne kvadratet af de første 10 naturlige tal, kan vi bruge en for loop sammen med en liste

```
kvadraterne = []
for tal in range(10):
    kvadraterne.append(tal ** 2)
print(kvadraterne)
>>> [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Opgaver til lister

- **Opgave 5.6.1:**
  Lav en liste med kvadraterne af tallene $1, 2, 3, ..., 100$ og print resultatet.

●●● **Opgave 5.6.2:**
Lav en liste hvor du bruger **en** for-loop til at finde differensen for hvert element i `kvadraterne` med hvert element i `kubik_tallene` elementvis (træk kvaradratet fra kubik-tallet). Print derefter resultatet.

```
# denne måde at definere lister på hedder list-
    ↪ comprehension, de nysgerrige kan slå det op på
    ↪ Google :)
kvadraterne = [i ** 2 for i in range(100)]
kubik_tallene = [i ** 3 for i in range(100)]
# Skriv din kode her
```

●●●● **Opgave 5.6.3:**
Konstrurere listen `[[[[[[[[[[’Hej’]]]]]]]]]]` ved at bruge en for loop. Start fra listen `[’Hej’]` og forsøg at lave flere ’lag’ med en for loop. Overveje derefter hvordan du ville gøre det for vilkårlige mange lag.

●●●●●●● **Opgave 5.6.4:**
Von-Neumann konstruktionen af de naturlige tal er den formelle måde indenfor ZFC hvorved de naturlige tal kan blive konstrureret, vi definerer de naturlige tal fx ved

$$\begin{aligned} 0 &:= \{\} \\ 1 &:= \{\{\}\} \\ 2 &:= \{\{\}, \{\{\}\}\} \\ 3 &:= \{\{\}, \{\{\}\}, \{\{\}, \{\{\}\}\}\} \\ &\vdots \end{aligned}$$

hvor $\{\}$ er den tomme mængde. Erstat nu mængder med lister, konstrurere de naturlige tal op til n vha. en for loop. Hint: De naturlige tal er rekursivt defineret ved gentagene brug af successor funktionen $S(n) = n \cup \{n\}$.

Dictionaries

Dictionaries kan man tænke lidt som en telefonbog, i gamle dage før internettet eksisterede brugte man telefonbøger til at slå op telefonnummrene på fx tømre eller eletrikere. Hvis man kender navnet på det selsskab man ledte efter, kunne man se at telefonbogen var organiseret i alfabetisk rækkefølge, og derefter slå deres nummer op. En dictionary på dansk kaldes for en ordbog, som også giver lidt et intuitivt billede af hvad de kan.

```
telefon_bogen = {
    "Jin": 56723822, # det her er ikke et rigtigt nummer
    "Louie": 66733234, # det her er ikke et rigtigt nummer
    "Politiet": 112 # det her tilgængeld er rigtigt
}
```

Man definerer en dictionary ved at skrive `{}`. Dictionaries har følgende struktur

```
min_dictionary = {
    key: value,
    key2: value2,
    key3: value3, # der skal tilføjes komma efter hver entry
    ... # man kan have vilkårlige mange entries (indgange)
}
```

Hvor `key` er typisk `int`, `str` og `value` kan være hvad som helst (`key` skal være en hashable type, slå det op på Google hvad det betyder). Vi kan så slå ting op i vores dictionary, fx hvis vi vil gerne have Louies nummer, kan vi skrive

```
louies_nr = telefon_bogen["Louie"]
print(louies_nr)
>>> 66733234
```

Vi kan tilføje en ny værdi til denne dictionary ved at skrive

```
telefon_bogen["Dig"] = 6213812
print(telefon_bogen)
>>> {'Jin': 56723822, 'Louie': 66733234, 'Politiet': 112, '
    ↪ Dig': 6213812}
```

Vi kan have mere avancerede dictionaries, fx kan vi have dictionaries i dictionaries

```
folk_fra_unf = {
    "Jin": {"tlf": 56723822, "hobbyer": ["programmering", "
       ↪ minecraft"], "alder": 19},
    "Louie": {"tlf": 66733234, "hobbyer": ["kompetitiv
       ↪ programmering", "unf", "fysik"], "alder": 20},
    "Dig": {"tlf": 6213812, "hobbyer": ["datalogi", "unf"],
       ↪ "alder": 420},
}
```

Vi kan så tilgå nogle af disse værdier ved at skrive:

```
print("Jins telefon nr. er", folk_fra_unf["Jin"]["tlf"])
>>> Jins telefon nr. er 56723822
print("Hans hobbyer er", folk_fra_unf["Jin"]["hobbyer"])
>>> Hans hobbyer er ['programmering', 'minecraft']
```

Vi kan også loope i gennem en dictionary, her looper vi med to variabler `navn` og `info` på `folk_fra_unf.items()` som giver os alle de key, value par for hver entry

```
# Prøv at køre koden i Colab!
for navn, info in folk_fra_unf.items():
    print("Vi har fat i", navn)
    print("Her er deres info", info)
```

Opgaver til dictionaries

●● **Opgave 5.6.5:**
Forsøg at printe informationerne i for-loop'en på en lidt pænere måde, gerne fx printe hver hobby for sig i stedet for at printe hele listen. Hint: Lav endnu en loop i for-loop'en for at printe listen af hobbyer

●●●● **Opgave 5.6.6:**
Lav et program der simulerer dictionaries vha. lister. Man skal kunne tilgå entries, appende, og fjerne elementer. Lav det hele uden at bruge Pythons dictionaries. Hint: tænk på en liste af lister med 2 elementer, kan man bruge dem til at konstrurere en dictionary?

●●●●●●● **Opgave 5.6.7:**
Lav en dictionary med alle de naturlige tal op til 15 som keys, og deres Von-Neumman konstruktioner som values. Det er måske en god ide at lave opgaven fra list-sektionen med at konstrurere de naturlige tal først.

Funktioner

Funktioner er nok en af de allervigtigste begreber inden for programmering, det kan være svært at forstå hvad en funktion er i starten, men den er et helt kritisk værktøj i programmørens værkstøjskasse som enhver programmør bør mestre brugen af. Det er nemmest i første omgang at tænke funktionen som en maskine, der gør et andet. Man

kalder funktionen, betyder at man tænder for maskinen. Vi har hidtil støt på `print`, som er en funktion der tager en string som *input* og printer den til konsollen. De fleste funktioner tager i mod en eller flere *input*s, og *returnerer* en *output*. Lad os definerer en simpel funktion

```
def kvadradtet_af(x):
    kvadrat = x ** 2
    return kvadrat
```

Funktionen oven kvadrere et tal, i dette tilfælde tager den i mod en input, et tal `x` og sætter den i anden, hvorefter værdien bliver tildelt til variablen `kvadrat`, som vi til allersidst retunerer. Vi kalder funktionen ved at skrive

```
ni = kvadradtet_af(3)
fire_treds = kvadradtet_af(8)
print(ni, fire_treds)
>>> 9 64
```

I koden oven finder vi kvadratet af tallene 3 og 8. Tallene erstatter `x` i definitionen af vores funktion, og derefter bliver `x ** 2` udregnet og så returneret. Vi kan også definere funktioner med flere input variabler, navnene på input variablerne er vilkårligt valgt

```
def sum_af_to_tal(x, y):
    summen = x + y
    return summen

# Man kan også skrive
def sum_af_to_tal(tal1, tal2):
    summen = tal1 + tal2
    return summen
```

Vi behøver egentlig ikke returnere noget fra funktionen, fx kan vi godt have at funktionen blot printer en string, i så fald er en `return` unøvendigt

```
def hils(unfer):
    print("Hej med dig", unfer, "godt at møde dig!")

hils("Louie")
>>> Hej med dig Louie godt at møde dig!
```

Typisk bruger man funktioner til at mindske den mængde af kode man skal skrive, hvis man fx har noget kode som gentages under forskellige omstændigheder, så giver det ikke mening at man skriver det samme kode flere gange hvis man allerede skrevet koden en gang. Programmering handler om at man ikke skal skrive mere end nødvendigt, grunden til at man fx har for loops er således at man ikke ender med at have kode såsom dette...

```
# Ser grimt ud...
print(kvadratet_af(0))
print(kvadratet_af(1))
print(kvadratet_af(2))
print(kvadratet_af(3))
print(kvadratet_af(4))
print(kvadratet_af(5))

# Lækker og kort
for i in range(6)
    print(kvadratet_af(i))
```

For at demonstrere hvorfor det er smart med funktioner, så kan det være at man bliver bedt om at anvende kalde funktioner på nogle elementer i en liste på en bestemt en

rækkefølge. Det kan være at man vil kvadrere alle tallene i en liste, derefter gange 5 og trække 2 fra, men hvad hvis man vil også prøve først at trække 2 fra, derefter kvadrere og gange med 5? Vi kan definere disse funktioner

```
def træk_2_fra(x):
    return x-2 # Man behøver ikke at definere en variabel
       ↪ for det man returnere

def gange_med_5(x):
    return x*5

def kvadrere(x):
    return x**2

def vores_funktion(liste_af_tal, liste_af_funktioner):
    resultat_listen = []
    for tal in liste_af_tal:
        resultat = tal
        for funktion in liste_af_funktioner:
            if funktion == 'kvadrere':
                resultat = kvadrere(resultat)
            elif funktion == 'gange5':
                resultat = gange_med_5(resultat)
            elif funktion == 'træk2':
                resultat = træk_2_fra(resultat)
        resultat_listen.append(resultat)
    return resultat_listen
vores_funktion([1, 2, 3, 4, 5], ["kvadrere", "gange5", "træ
   ↪ k2"])
>>> [3, 18, 43, 78, 123]

vores_funktion([1, 2, 3, 4, 5], ["træk2", "kvadrere", "
   ↪ gange5"])
>>> [5, 0, 5, 20, 45]

vores_funktion([1, 2, 3, 4, 5], ["kvadrere", "gange5", "træ
   ↪ k2", "træk2"])
>>> [1, 16, 41, 76, 121]

vores_funktion([1, 2, 3, 4, 5], ["kvadrere", "gange5", "træ
   ↪ k2", "træk2", "træk2"])
>>> [-1, 14, 39, 74, 119]

vores_funktion([1, 2, 3, 4, 5], ["gange5", "gange5"])
>>> [25, 50, 75, 100, 125]
```

Der er flere sjove ting man kan lave med funktioner, fx kan man lave en funktion der returnerer en funktion, eller rekursive funktioner som I kommer til at se i den næste kapitel.

Opgaver til funktioner

●●● **Opgave 5.6.8:**
I Python kan man lave en liste af funktioner, fx i koden tidligere kunne man have defineret listen `liste_af_funktioner = [kvadrere, gange_med_5, træk_2_fra]`, man kan så kvadrere fx ved at skrive `liste_af_funktioner[0](3)`. Brug en liste af funktioner til at erstatte if-udtrykkene i `vores_funktion`.

●●●● **Opgave 5.6.9:**
Lav en funktion der returnerer det inderste element givet ved funktionen

```
def wrap(tal, gange):
    wrapped = [tal]
    for i in range(gange):
        wrapped = [wrapped]
    return wrapped

# fx kan vi kalde
hvad = wrap(5, 100)
print(hvad)
>>> [[...[[[5]]]...]] # ... repræsenterer mange []
    ↪ parenteser
```

Hint: Man kan tjekke om hvis noget er en liste ved at skrive `type(x)== list`, brug denne information samt en while-loop

●●●●●● **Opgave 5.6.10:**
En **matematisk** funktion $f : X \to Y$ er defineret som en relation mellem to mængder X og Y. Helt formelt kan man definere en matematisk funktion, f, som en mængde af ordnede par, hvorved der gælder det følgende udsagn:

$$\forall x \in X \forall y \in Y \forall z \in Y : (x, y) \in f \land (x, z) \in f \implies y = z.$$

Dvs. for ethvert input x til funktionen, hvis det gælder at $f(x) = y$ og samtidigt $f(x) = z$ så må $y = z$.
Givet at du nu har modtaget en liste af ordnede par i Python (et ordnet par kan tænkes som en liste med 2 elementer): `[[1,2], [4, 5], [2, 3], [6, 3]]`, så skal du skrive en funktion i Python der returnerer enten en `True` eller `False` afhængigt af hvorvidt listen kan udgør definitionen for en funktion. Antag at et ordnet par ikke kan optræde flere gange i listen. Hint 1: start med at skrive `def er_en_funktion(f):` hvor `f` er den liste du får. Hint 2: Prøv at slå op Pythons indbygget funktioner `set()` og `len()` og overvej hvordan du kan bruge disse til at bekræfte hvorvidt `f` er en funktion.

●●●●●●● **Opgave 5.6.11:**
Lav en funktion der giver dig Von-Neumann konstruktionen for et tal n.

5.7 Algoritmer og biblioteker

I dette kapitel vil vi beskæftige os med algoritmer. Hvad er en algoritme, er det første spørgsmål man bør stille sig selv, når man møder ordet. Forsimplet, så er en algoritme, en sekvens af operationer, der udføres på et input, og derefter terminerer. Læg her mærk til det sidste ord, terminerer. Det er meget vigtigt, at algoritmen terminerer, ellers kan vi aldrig vide om den rent faktisk virker. En algoritme kunne være, en funktion der tager et vilkårligt antal tal og lægger dem sammen, også kaldet en sum. Vi vil i de næste delkapitler kigge på nogle simple algoritmer, som kan bruges til en lang række problemer.

Recursion

Vi starter ud med at diskutere rekursion, som er en bestemt type af algoritme. Rekursion er når en værdi i en mængde afhænger af en eller flere værdier i selvsamme mængde. Man kan for eksempel definere de naturlige tal rekursivt:

$$x_n = x_{n-1} + 1$$

Hvor n går fra $1, 2, ...$ til uendeligt, notationen x_n betyder "det n'te x'er", og vi kalder n for indekset. Vi mangler dog en betingelse for at færdiggøre den rekursive definition. Prøv at tænk over hvad den skal være, inden du læser videre.
For at den rekursive definition er veldefineret, skal man have en startværdi. I tilfældet for værdien af et naturligt tal, vil det være

$$x_1 = 1$$

I Python kan man skrive ovenstående, som et stykke kode.

```
def vaerdi_naturlige_tal(n):
    if n == 1:
        return 1
    else:
        return 1+vaerdi_naturlige_tal(n-1) #Kig tilbage på
            ↪ rekursionsligningen og overbevis dig selv om,
            ↪ at det samme står her i koden.
```

Det her if statement indeholder det vi kalder en *basecase*. En basecase eller startværdi, er noget man ved om nogle af de første værdier i følgen.

●●● **Opgave 5.7.1:**
I stedet for at finde det n'ete naturlige tal (hvilket jo var ret åbenlyst), så vil vi nu forsøge at finde summen af de naturlige af de naturlige tal fra 1 til og med n. Hvis nu n er 5, vil summen være 5+4+3+2+1. Startværdien er her $sum_1 = 1$. Det rekursive udtryk er at $sum_n = sum_{n-1} + n$. Prøv nu at implementer en funktion i python der kan regne ovenstående rekursionsligning.

●●● **Opgave 5.7.2:**
I denne opgave skal du arbejde med fakultetsfunktionen. Fakultet, som skrives $n!$, hvor n er et helt tal større end nul er beskrevet på følgende vis. $n! = n \cdot (n-1)!$ og startværdien er $0! = 1$. Du har fået givet startværdien/basecasen samt det rekursive udtryk. Implementer nu en funktion i Python der kan regne n! ud.

●●●● **Opgave 5.7.3:**
Denne opgave beskæftiger sig med potenser. Vi vil gerne vide, hvad 3^n er, hvor $n \geq 0$. Du må her ikke bruge din viden om at Python bare kan udregne resultatet for dig.

1. Prøv at tænke over hvad en basecase eller startværdi kunne være. Hint: hvad er 3^0?
2. Du har nu fundet frem til hvad basecasen er. Kan du skrive et udtryk op for x_n, som afhænger af x_{n-1}
3. Du har nu skrevet et udtryk op. Implementer det i Python.
4. Ekstra opgave: Generaliser din funktion, så du kan regne x^n

●●●●● **Opgave 5.7.4:**
Fibonacci tallene er en sekvens af tal, som er spændende indenfor naturvidenskab, da forholdet mellem to sideliggende værdier konvergerer mod den gyldne ratio. Den gyldne ratio ses ofte i naturen. Fibonacci sekvensen er defineret, som

$$fib_n = fib_{n-1} + fib_{n-2}$$

Hvor at $fib_1 = 0$ og $fib_1 = 1$ Du skal nu finde det n'te fibonaccital med en funktion i python.

Sortering

Nu går vi videre til en af de vigtigste ting inden for at arbejde med datastrukturer, nemlig sortering. At sortere en mængde, data er essentielt. Vi kan let finde medianen i et sorteret datasæt, max og min værdier samt meget andet. I dette afsnit vil vi kigge på en sorteringsalgoritme nemlig insertion sort. Måden hvorpå insertion sort fungerer er:

- Itererer fra det andet element til det sidste element i listen
- Hver gang du er ved et element sammenligner du det forrige element med det nuværende.
- Hvis det forhenværende element er større en det nuværende, så byt rundt på elementet og det nuværende. Nu vil dit nuværende element være på det $i - 1$ elements plads. Fortsæt dette skridt indtil at elementet er større en elementet forinden eller at elementet er blevet placeret på den aller første plads.

●● **Opgave 5.7.5:**
Givet et array med 3 elementer. Find på hvad de tre elementers værdier kan være for at du skal lave et maksimalt antal sammenligner og iterationer for at sortere de tre elementer.

●●●●● **Opgave 5.7.6:**
Kan du sige noget mere generelt om hvor mange sammenligninger og iterationer man skal lave i det værste tilfælde. Hvad med i det bedste tilfælde?

●●●●●●●●●● **Opgave 5.7.7:**
Denne opgave er til de ekstra nysgerrige. Prøv selv at implementere insertion sort som skrevet foroven. Vink: brug et for loop til at iterere over elementerne og under hver iteration brug et while loop for at lave sammenligningerne og rokaderne af elementerne.

Fremadrettet må I gerne bruge Pythons `sort` funktion.

●●●● **Opgave 5.7.8:**
Find medianen af datasættet `[-1, 5, -5, -9, 1, -17, 3, 10, -3, 100, 73, 64, 902, 123, 653]` Husk: medianen er givet som det midterste element hvis der er et ulige antal elementer og ved et lige antal elementer er gennemsnittet af de to midterste elementers værdier.

●●●●● **Opgave 5.7.9:**
Jin har N tallerkner, og han vil gerne stable dem hurtigst muligt. Jin vil gerne stable sine tallerkner ved først at stable de to tallerkner, som er tættest på hinanden og derefter fortsætte denne process. Du skal i dette tilfælde kun hjælpe Jin med at finde de første to tallerkner, som han skal lægge ovenpå hinanden. Du er givet en liste af punkter og skal ud fra denne liste bestemme afstanden de to tallerkner som ligger tættest på hinanden. `[173.47, 1028.42, 479.5, 127.82, 802, 386.5, -102, 2034.2, 17.3, 640]`. Opgaven kan løses på flere forskellige måder.

Two pointers

En metode som kan benyttes til at løse rigtig mange problemer effektivt er two pointer metoden. I two pointer metoden holder man konstant styr på to af sine elementer og deres værdier og bruger disse oplysninger smart.

Forestil jer at vi står på i begyndelsen af en endelig gang, hvor der er lokaler på venstre side. Vi ved ikke hvor mange lokaler der i alt, men vi har fået af vide af vores klasselærer, at vi skal finde det midterste lokale. Vi kunne godt bare tælle alle lokalerne, men vi kan

faktisk gøre noget andet som er super smart. Vi er to personer, og vi lægger den strategi, at hver gang jeg passerer et lokale, så skal du passere to lokaler. Når du så når det sidste lokale så gangen råber du tilbage til mig "Der er nu ikke flere lokaler. Du har fundet det midterste". Kan du se hvorfor dette er sandt?

Ideen er her, at det midterste lokale nødvendigvis antallet er lokaler divideret med to. Fordi du går to lokaler frem ad gangen, og jeg kun et, vil jeg have gået halvt så langt som dig, når du har nået enden af gangen. Derfor er jeg altså ved det midterste lokale.

Vi vil nu opstille et eksempel, denne gang med kode, hvor two pointers teknikken bruges. Vi er givet et sorteret array af tal, og vi har fået at vide, at to af tallene lagt sammen er lig x. Da tallene er sorterede ved vi at den mindste sum må være det første element lagt sammen med det andet element, og at den største sum må være når vi lægger den sidste værdi sammen med den næstsidste. Vi kan derfor starte med at kigge på summen af det første og det sidste element. Hvis denne sum er mindre end x, så vil vi gerne have en større sum. Dette kan vi nu kun gøre ved at tage og element det andet element sammen med det sidste. Hvis nu summen er for stor, så skal vi nu gøre summen mindre. Dette kan gøres ved at vælge det næst sidste element og lægge det sammen med det andet element. Denne process fortsættes indtil, at man finder en sum som er lig x.

```
arr = [1, 3, 18, 23, 103, 527, 700, 1903, 12039]
a = 0
b = len(arr)-1 #Lister er jo nul indekserede
x = 550
while a <= b:
    summen = arr[a]+arr[b]
    if summen == x:
        break
    elif arr[a]+arr[b] > x:
        b -= 1
    else:
        a += 1

print(f"De to tal der lagt sammen gav {x} var {arr[a]} og {
    ↪ arr[b]}")
```

●●●●●● **Opgave 5.7.10:**
Find den mindste sum af to tal i en liste, som er større end x. Vink: prøv at tag inspiration fra den kode du ser foroven.

●●●●●●●● **Opgave 5.7.11:**
En delsum er defineret som at man tager elementerne fra et index til et andet index og lægger disse værdier sammen. Givet en liste af tal find den maksimale delsum.

●●●●●●●● **Opgave 5.7.12:**
En delsum er defineret som at man tager elementerne fra et index til et andet index og lægger disse værdier sammen. Givet en liste af tal find den maksimale delsum.

●●●●●●●●●● **Opgave 5.7.13:**
Roter elementerne i et array k gange. Med rotation menes, at man rykker elementerne en til højre, og hvis elementet er det sidste element rykkes det til start.

Biblioteker

I Python kan man importere kode som andre har skrevet for at gøre ens liv lettere, man kalder disse kode for "biblioteker". Der findes masser af Python biblioteker hvis man

googler på internettet, lad os kigge på de biblioteker der hedder `math` og `random`. Du kan importere et bibliotek ved at skrive:

```
import random
```

Her vil random biblioteket blive importeret, og den har en række funktioner som vi kan kalde. Google evt. "random library python" for at se hvilke funktioner den har. Som et eksempel kan vi genererer et tilfældet decimaltal mellem 0 og 1 ved at skrive:

```
import random

tilfældigt_tal = random.random()
print(tilfældigt_tal)
>>> 0.32935699857938416
```

Med `math` biblioteket kan vi kalde matematiske funktioner

```
import math

print(math.pi) # Konstanten pi
>>> 3.141592653589793

print(math.exp(1)) # Konstanten e
>>> 2.718281828459045

print(math.factorial(6)) # Beregner 6!
>>> 720
```

Når vi importerer et bibliotek skriver vi som regel `import bibliotek`, derefter for at kalde bibliotekets funktioner kan vi skrive `bibliotek.funktion()`. Vi kan også importere specifikke funktioner fra biblioteket ved at skrive `from bibliotek import funktion`, så behøver vi ikke at kalde funktionen ved at skrive `bibliotek.` foran. Fx kan vi så skrive:

```
from random import random

print(random())
>>> 0.6595022595617526
```

5.8 Er datalogi en naturvidenskab?

Datalogi i sig selv kan ikke helt kaldes for en naturvidenskab, datalogiens ontologiske status bygger på matematikkens, og især indenfor klassisk datalogi (vi kigger bort fra kvantekomputere), så opnås ny viden i datalogi ikke gennem den naturvidenskabelige metode, men i stedet gennem det aksiomatisk metode. Resultater vedrørende algoritmer og køretider i datalogi bliver bevist vha. matematiske metoder, og er fundamentalt anderledes end fx fysik, kemi eller biologi, hvor man bl.a. anvender induktive metoder ved at indsamle mange observationer. Til gengæld, anvendes den naturvidenskabelige metode i et meget specifikt område indefor programmering, og det kan argumenteres, at det er en af reneste anvendelser af den naturvidenskabelige metode, eller i hvertfald den deduktive metode.

Den naturvidenskabelige debugging

Debugging, eller fejlfinding, er det vigtigste evne for alle programmører at mestre. Det er et kendt ordsprog blandt programmører at kun 20% af den tid man bruger i udvikling, skriver man rent faktisk kode, de resterende 80% af tiden er brugt på at fejlfinde. Processen i at fejlfinde, er en ren naturvidenskabelig proces. Det handler om at man vil

gerne identificere en *årsag* til et *symptom* som man *observerer* i ens program, hvilket kan gøres vh.a. den deduktive metode. Som programmør opstiller man så en række *hypoteser* som man tester af, i håbet om at finde hvad symptomet egentlig skyldes.

Identificering af symptomet

Symptomet, eller fejlen, er noget der ikke stemmer overens med vores forventning om hvordan vores program bør have eksekveret. Symptomet skal præciseres i sprog før man er i stand til at anvende den naturvidenskabelige metode, fx er følgende eksempler på en god identificering af symptomet vs dårlige identificeringer. Her er der taget udgangspunkt i at man fx har en funktion der ikke returnerer det rigtige output:

1. Dårlig identificering, denne er en hypotese, ikke en identificering: "Min funktion returnerer ikke den rigtige output fordi jeg ikke kørte den med de rigtige parametre"

2. Dårlig identificering, vagt udsagn: "Programmet laver ikke det den skal"

3. God identificering: "Funktionen returnerer en `None` value, hvor den skulle have returneret en liste med 5 tal.

Man skal præcisere hvad man mener når man siger at ens program ikke virker, på hvilken måde burde det have virket? Kom med tekniske detaljer på hvad der skulle have returneres og hvor henne. En identificering af symptomet kan tænkes som det man læser direkte fra skærmen, uden at man har gjort nogen tanker omkring hvad det kan skyldes.

Hypotese-dannelse

En god identificering af symptomet gør typisk at man meget hurtigt kan danne sig nogle hypoteser som man vil forsøge at teste af. Man skal desuden være meget opmærksom på de antagelser som man har gjort sig for at have nået til denne fase. Det er vigtigt, at man ikke allerede i starten da man skulle identificere symptomet, laver for mange antagelser, for så vil man kunne risikere at bruge lang tid på at jagte efter en falsk hypotese, som man ikke ville have gjort hvis man har identificeret symptomet klart og tydeligt. Fx kunne et eksempel være at man tror at man har kaldt en specifik funktion før man kørte ens kode, det er en antagelse som kunne være forkert, og i lyset af at man har antaget noget forkert, kommer ens hypoteser heller ikke til at være baseret i virkeligheden.

Selve hypotese-dannelsen kommer så efter man har identificeret symptomet. Vi kan kalde vores symptom for p som er blot et udsagn eller påstand, fx fra tidligere eksempel:

$$p : \text{Funktionen returnerer en } \texttt{None} \text{ value}$$

Derefter er vores arbejde at forsøge om at finde årsagen q hvor det gælder at $q \implies p$, som læses "q medfører p". Et simpelt eksempel på q kunne være:

$$q : \texttt{return} \text{ statement var blevet kommenteret ud ud}$$

Som kan verificeres hvis man læser det sted i koden hvor man har ens `return` statement. Men det er ikke nok at verificere q, vi skal også vide at den q som vi har fundet, rent faktisk forårsager vores symptom. Her benytter man en kendt slutningsregel, *modus ponens*, som siger at vi skal vide at q eksisterer, og samtidigt at $q \implies p$ vi kan sige at q er den rigtige årsag. Det kan være en af årsagerne, der kan også eksistere flere andre årsager som potentielt kan resultere i det symptom vi observerer, altså kan der være tale om en række årsager, $q_1, q_2, ..., q_n$, der hver især medfører p. Vores job er så at udelukke alle de årsager som ikke findes, dvs. vi skal konstatere at de ikke er tilstede i vores program, her kan være smart at benytte *kontraponering* som kommer fra *bevis ved kontraposition*. Typisk har flere af disse mulige årsager $q_1, q_2, ..., q_n$ også visse "side-effects", $s_1, s_2, ..., s_k$, som er andre konsekvenser der kan ske udover det specifikke

symptom som vi observerer. Vi ved **på forhånd** (typisk gennem erfaring) at disse side-effects sker hvis q'erne sker. Disse side-effects kan være noget som vi meget nemt kan bekræfte eller afkræfte ved at observerer programmets opførelse. Vi vil så være sikker på at en årsag q_1 medfører en række side-effects: s_1, s_2, s_3, så kan vi blot forkaste en af disse (kommer an på hvad der er nemmest) for at modbevise q_1 findes, denne form for kontraponering af en implikation ser således ud i logisk form:

$$q \implies p \equiv \neg p \implies \neg q$$

Hvor symbolet $\neg$ betyder "ikke" eller "negation", og $\equiv$ betyder "ækvivalens", som i at de to slutningsformer er ækvivalente. Et konkret eksempel kunne være at q stod for "Det regner" og p var "Jorden er våd", implikationen $q \implies p$ har så den tolkning: "Hvis det regner, så bliver jorden våd". Ved kontraponering kan vi så slutte at $\neg p \implies \neg q$, hvilket betyder "Hvis jorden ikke er våd, så regner det ikke". Kontraponering er et af de stærkeste værktøjer man har når man skal fejlfinde.

Konkret debugging eksempel

Lad os tage udgangspunkt i det tidligere eksempel hvor man havde en funktion der returnerede et `None` frem for en liste af 5 tal. Vi har umiddelbart identificeret symptomet direkte ved at sige helt konkret på det kode-mæssigt niveau. Vi har nu muligheden for at opstille flere forskellige hypoteser for hvad er årsagerne til dette symptom, de hypotetiske årsager kunne fx være:

1. q_1: `return` udtrykket var blevet udkommenteret
2. q_2: Der var en stavefejl da vi kaldte funktionen med dens navn, altså var der en anden funktion der hed det samme
3. q_3: Der var sket en kvantemekanisk fluktuation i computerens hukommelse som slettede listen
4. q_4: Vi havde Chrome-browseren åbent samtidigt da vi kørte koden

Det ses at q_3 en sjov hypotetisk årsag, som ikke nødvendigvis er forkert, da den vil sagtens gør at listen ville blive slettet, men hypotesen er altså svært at bekræfte eller afkræfte hvilket gør at den ikke nødvendigvis er en særlig god hypotese. Man kunne så sige, at sandsynligheden for at q_3 sker er ekstrem lille, måske hvis man derfor kørte funktionen en anden gang og den så lykkedes, så ville man have noget grundlag for at acceptere hypotesen, men det er stadigt meget tvivlsomt om hvis man ville det.

Vi ser at q_4 er nemt at teste for, dog allerede på forhånd ser vi at den er en lidt dum hypotese da det slet ikke klart er hvordan ens Chrome-browser vil kunne påvirke den Python kode som vi kører. Der skal være en meget mystisk forklaring bag hypotesen for at gøre q_4 rent faktisk medfører vores p. Hypotesen kan nemt afkræftes hvis vi slukker Chrome-browseren og kører koden igen (eller kan vi det? Se en af opgaverne), og man burde egentlig ikke have dannet denne hypotese til at begynde med.

Til sidst ser man at q_1 og q_2 er eksempler på nogle mere fornuftige hypoteser, der er nemt at tjekke og som ville samtidigt kunne medføre vores p, hvis de rent faktisk var sande. Men lad os antage nu at vi har tjekket q_1 og q_2 ved at læse i koden, og vi ser at de ikke er tilstede, hvad kan vi så gøre? Vi bliver så nødt til at opstille andre forklaringer, fx kunne man spørger om hvis koden bliver rent faktisk gemt, altså kunne man så forsøge at modbevise at ens kode er gemt, så ville ens hypotese være

q : Koden er gemt

Vi ser at en umiddelbart konsekvens/side-effect kunne være

s : Alle `print` udtryk vi skriver ville blive printet

Så kunne en taktik for at modbevise q være at vi skrev nogle `print` udtryk i funktionen, hvis vi så observerer at de *ikke* bliver printet til konsollen, så ville vi kunne ved kontraponering konkludere at koden ikke blev gemt (eller i hvertfald ikke gemt ordentlig). Denne slutning kan vi lave fordi vi ved på forhånd at $p \implies s$, dermed betyder det at $\neg s \implies \neg p$. Vi vil måske så se at vi ikke har trykket på Ctrl-S for at gemme, og vi vil have god grund til at acceptere det som årsagen til vores symptom hvis vi ser, efter vi har gemt koden, at den kører som vi havde forventede.

Perspektivering af debugging mod andre naturvidenskab

Det kan derfor argumenteres at fejlfinding i programmering er nok den reneste anvendelse af den deduktive metode i naturvidenskabelige sammenhæng. Debugging er et fantastisk eksempel på det deduktive metode, fordi software er uafhængigt af tid, betyder det at hvis man kørte noget kode i dag, ville det ikke køre anderledes i morgen hvis ikke man har ændret i koden eller opdateret ens system. Denne tids-uafhængighed giver os muligheden for at eksperimentere med ændring til en variabel ad gangen, og gør at vi kan være ret sikker på at vi har fundet fejlen. I modsætning til andre naturvidenskab, kan det nogle gange være svært at sige hvad har forårsaget en begivenhed, da man ikke kan gå tilbage i tiden. Nogle gange har man kun en enkel observation til rådighed, hvilket gør at man kun må anvende *abduktion* eller slutning til bedste forklaring, som ikke er særligt tilfredsstillende.

Nu har vi kun nævnt den deduktive metode, men man benytter faktisk også induktion og abduktion i debugging processen. Abduktion, eller slutning til bedste forklaring, bruges når vi genererer vores hypoteser. Hypoteserne er valgt efter hvad der giver bedst mening, og her vil vi altid gerne identificere de mest *sandsynlige* årsager først. Induktion benytter vi når vi generaliserer fra de observationer vi har, det kan være meget nyttigt at identificere symptomet. Fx kunne vi godt observere en mærkelig opførelse blandt de funktioner der kalder en bestemt funktion X, så kan vi generalisere ved at sige at vores p er "Alle funktioner der kalder X har en mærkelig opførelse". Men vi skal passe på når vi benytter disse induktive argumenter, vi husker at vores symptom p er selv et udsagn, som godt kan være forkert. Det er derfor kritisk at vi er helt sikker på at vores identificering af symptomet er korrekt. For at teste at hvorvidt symptomet er sandt, kan vi betragte de side-effects som symptomet ville medføre, og se hvis de ikke er tilstede, og derefter benytte kontraponering for at modbevise at vores symptom var korrekt identificeret.

Tænke-opgaver til debugging og NV-metoden

●● **Opgave 5.8.1:**
Hvordan kan vi være sikker på at $q \implies s$, hvis vi tager udgangspunkt i det konkrete eksempel fra tidligere. Kan vi altid være sikker på at en årsag har den konsekvens som den har? Overvej hvad der ligger grund for denne antagelse.

●● **Opgave 5.8.2:**
Kan vi slutte $p \implies q$ fra $q \implies p$? Overvej hvis q hed "Det regner" og p hed "Jorden er våd". Konkluder hvorfor vi ikke nødvendigvis kan slutte årsagen direkte efter at have observeret en side-effect.

●●● **Opgave 5.8.3:**
Kontraponering gør os i stand til at identificere hvilke årsager **ikke** kan være rigtige, overvej om hvis der findes fordele ved at benytte kontraponering, frem for at forsøge at pege på en årsag direkte. Overvej det i sammenhæng med at et symptom eller side-effect kan være resulteret på grund af flere *mulige* årsager.

●●●● **Opgave 5.8.4:**
Med udgangspunkt i det konkrete debugging eksempel fra før, hvis vi observerer at koden ikke kører når vi har Chrome-browseren åbent, og observerer samtidigt at den heller ikke kører når vi lukker for browseren, er der så grundlag for at konkludere at

årsagen ikke skyldes Chrome-browseren? Tænk q: "Chrome-browseren er åbent", hvis p sker under både q og $\neg q$, er p så uafhængigt af q eller $\neg q$?

●●●●● **Opgave 5.8.5:**
Hvis vi efter at have trykket på Ctrl-S for at gemme, finder ud af at vores kode rent faktisk kørte, har vi så nok grundlag til at sige at årsagen var at vi ikke gemte koden? Prøv at google "underdetermination" og læs Wikipedia artiklen, overvej hvordan begrebet kan være relevant i denne sammenhæng. Giv derefter et eksempel på en anden årsag (som gerne må være absurd), der potentielt kunne have i stedet gjort at vores kode virkede.

●●●●●●● **Opgave 5.8.6:**
Hvordan skal vi vælge de hypoteser vi har lyst til at teste? Kan du tænke på nogle regler/tips for hvad der definerer gode hypoteser? Tænk: hvor mange hypoteser kan man egentlig lave?

5.9 Projekter

Nu har I lært hvordan Python fungerer, så kunne det være sjovt at anvende jeres programmeringsviden i nogle projekter som vi har lavet, her kan I vælge mellem to forskellige projekter:

Projekt i at approksimere π

Konstanten π dukker op overalt, og ikke mindst kan vi ved brug af numeriske metoder, forsøge at tilnærme os konstanten. For dem der ikke ved hvad π er, så tænk på runde objekter I har set i virkeligheden. Hvis man tager et rundt objekt, måler dets omkreds og dividerer med dets diameter, så ville man se at uanset hvilket objekt man tager, er forholdet cirka lig $3.14 \approx \pi$. Vi kan så forestille os at hvis vi havde det perfekte runde objekt der havde perfekt lige omkreds, så kan vi tilnærme os π meget præcist med en meget præcis lineal.

I dette projekt tager vi en anderledes tilgang til at approksimere π, vi vil benytte os metoden Monte-Carlo, som handler om at man vh.a. tilfældighed (tilfældige tal), kan lave nogle rimelig præcise approksimationer. I bedes om at gå ind på `https://bit.ly/3HqDyZE`, som er en Google Colab notebook hvor vi har nogle opgaver klar til jer.

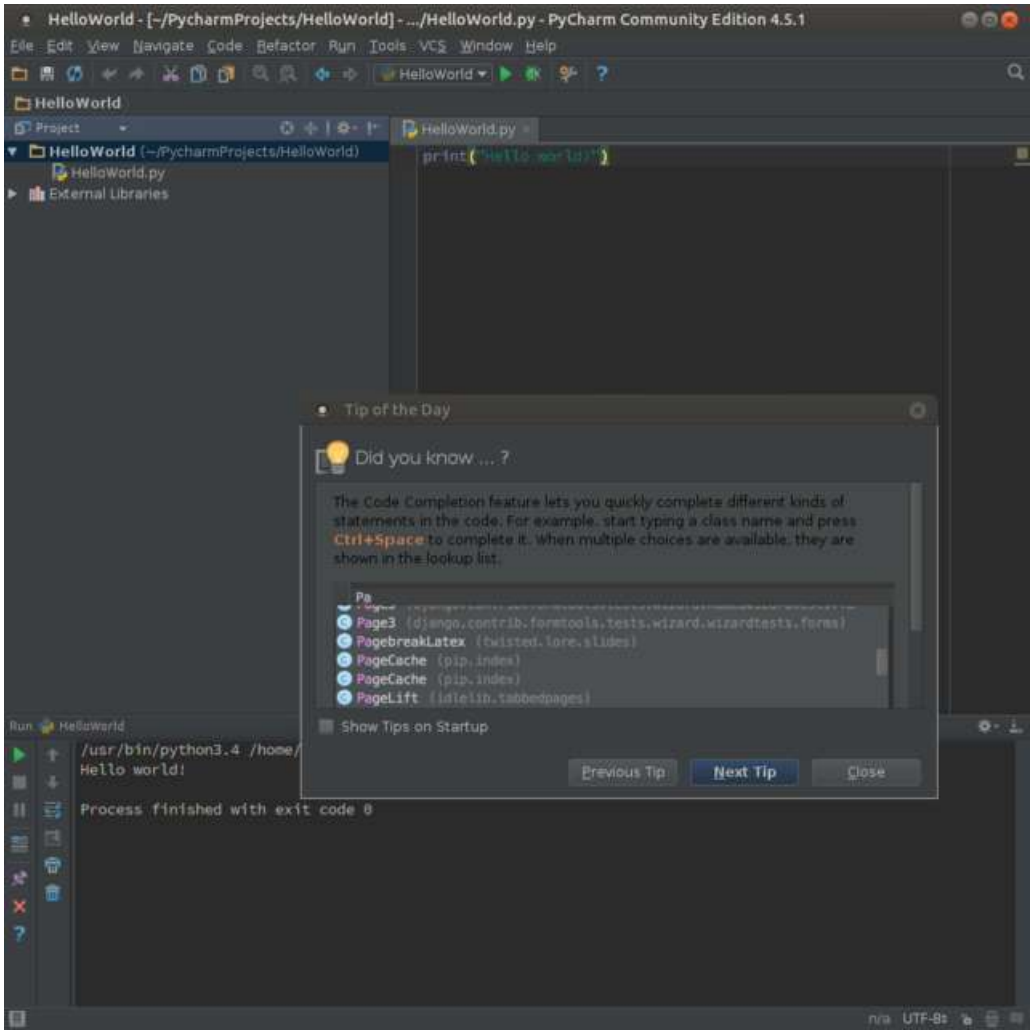
Rekursions projekt

5.10 Konklusion og viderebygning

I har nu været igennem en del af datalogiens verden, en meget meget lille del. Der er meget meget mere at lære, og man kan gå i mange forskellige retninger. Så dette afsnit er dedikeret til forskellige retninger og tilhørende ressourcer, så kan I selv kigge dem i gennem.

Googling og StackOverflow

Google, google og google. Det kan ikke understreges nok hvor vigtigt det er at kunne google som programmør, google alt det som I ikke forstå. Der er så mange detaljer vi har undladt at skrive om i dette afsnit for Datalogi, simpelthen fordi det ikke kan lade sig gøres at dække alt omkring programmering, det er derfor vigtigt at I selv finder frem til de kilder ved at google. En fejl som mange typisk laver er at google på dansk, det er svært at finde brugbare resultater hvis man googler på dansk, oversæt dit spørgsmål til engelsk i stedet når du forsøger at finde noget, for ellers bliver du skuffet. Typisk kan man finde svar til ens programmeringsspørgsmål på en hjemmeside der hedder StackOverflow,



Figur 5.4: PyCharm CE (Screenshot er taget fra Wikimedia Commons, public domain)

det er et forum hvor man kan stille spørgsmål til (typisk) mere erfarende programmører (google det :)).

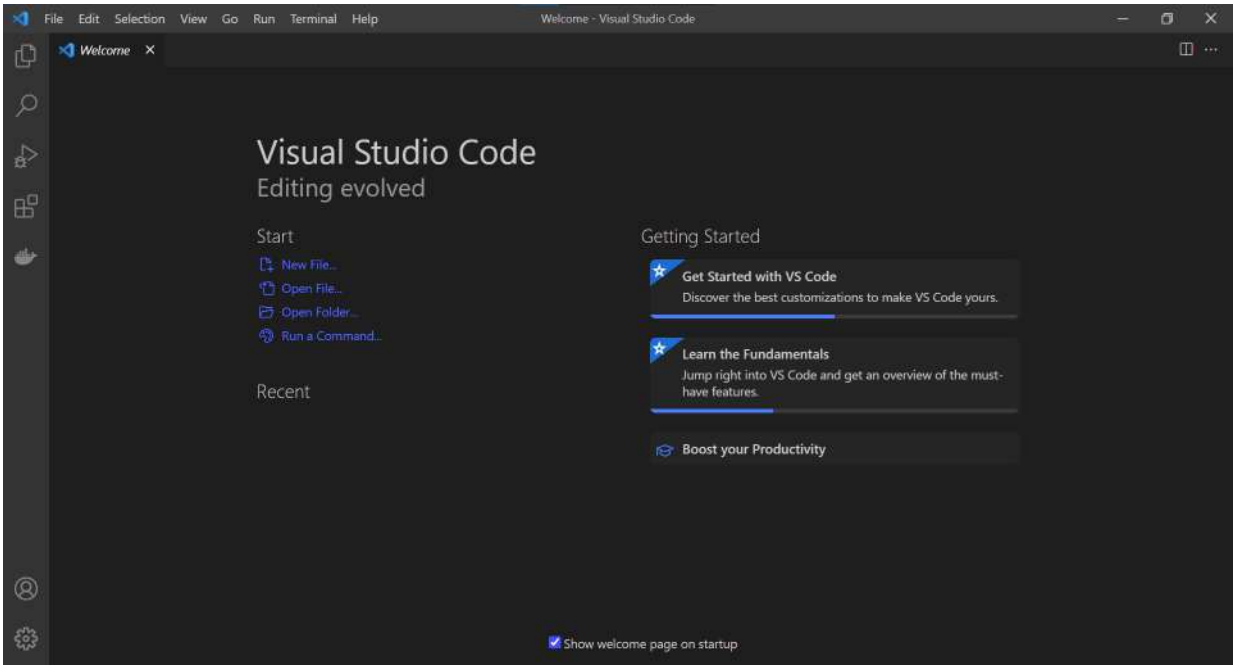
Editors og IDEs

Hvis man vil gerne lave større Python projekter, eller bare kode projekter generelt, så kan det være en god ide at sætte en kode editor op. En kode editor eller IDE som står for Integrated Development Environment, kommer med adskillige features der gør det nemmere for programmører at spotte fejl i deres kode. Vi anbefaler bl.a. PyCharm og Visual Studio, som er nok de mest brugte IDEs.

Andre programmeringssprog

Der findes mange andre programmeringssprog i verden, I kan selv google jer frem til hvilke de er. Her er dog en liste af de mest populære programmeringssprog og deres beskrivelser:

- **Java**: Java er nok en af de mest kendte programmeringssprog, og var helt sikkert den mest brugte i starten af den 20 århundrede. Java programmering følger noget som man kalder for OOP (Object Oriented Programming) paradigmet, hvilket er en bestemt måde at tænke kode på. Det kan være utroligt nyttigt at forstå forskellene mellem programmeringsparadigmerne (google dem!). OOP paradigmet



Figur 5.5: Visual Studio Code (Screenshot er taget fra Wikimedia Commons, public domain)

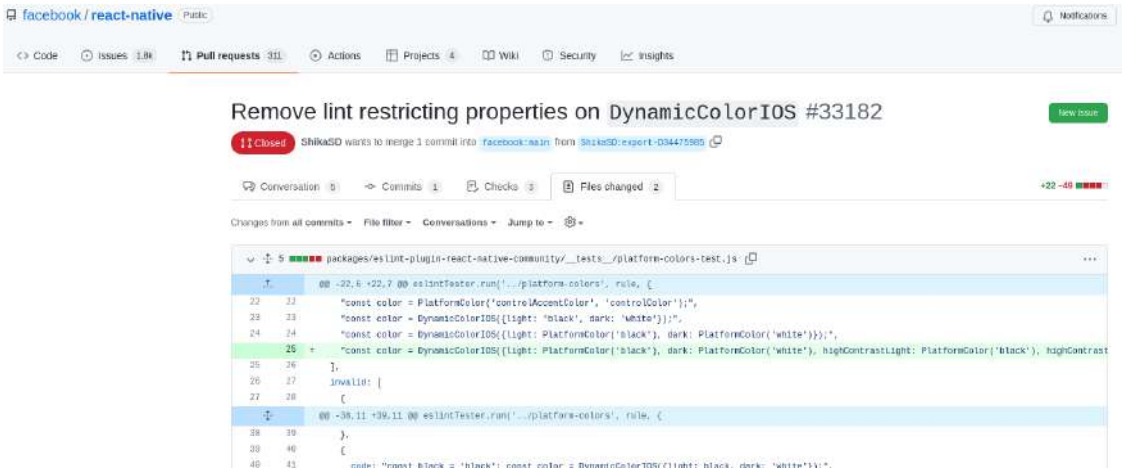
haft en kæmpe indflydelse på alt vi ser på internettet i dag. Mange nye startups i dag er gået væk fra at lave OOP til at lave funktionel programmering, og det er fx noget som JavaScript er kendt for. Java er typisk brugt i store coporate enterprise firmaer, dvs. banker, store administrative bureauer.

- **JavaScript**: Har næsten intet at gøre med Java, bare lige heads up. JavaScript er det der kører på ens browser når man loader en hjemmeside, altså bliver den brugt til at lave ret meget frontend-programmering som er programmering rettet mod bruger-vendt interfaces (google hvad det betyder). JavaScript bruges typisk i de meget populære frameworks som er Vue, React Native, Express.js (på node.js) (google!). Man siger at JavaScript er funktionel programmering, fordi man arbejder ikke så meget med at manipulere states på objekter, i modsætning til OOP programmering.

- **C og C++**: C er nok den allerførste moderne programmeringssprog, og C++ baserer sig på C som kommer med nogen flere features. Man skriver C eller C++ hvis man virkelig vil gerne optimere ens kode, og lad den køre hurtigt. Et program skrevet i Python, kan være op til 45 gange langsommere end det samme program skrevet i C (google "Python vs C speed"). Man kan så spørge, hvorfor skriver alle ikke bare C eller C++? Det er et godt spørgsmål, og svaret ligger i at hvis man selv forsøger at lære C eller C++, så indser man hvor svært det er at forstå noget C kode, fremfor at bare køre Python på en bedre computer. Ej, det er lidt overdrevet, det reelle svar er noget i den stil, der er ikke så mange biblioteker/værktøjer til C eller C++ i modsætning til Python. Python er meget mere nemmere at lære og meget mere overskueligt at læse. Man bruger dog C i mange steder hvor det kræver det kører hurtigt. Fx er de fleste styresystemer skrevet i C eller C++, og Python er faktisk selv skrevet i C!

- **PHP**: Lad vær med at bruge det her, bare kør med JavaScript og Express.js i stedet...

- **Assembly**: Hvis man vil helt ned til metallet, og arbejde med at flytte bits rundt i en computerens hukommelse, samt fortælle direkte til CPU'en hvad den skal gøre,



Figur 5.6: En GitHub pull request

så skriver man i Assembly language. Assembly er i sig selv ikke er et programmering sprog, men i stedet en kategori af programmeringssprog.

Programmering i virkeligheden

Programmering i den virkelige verden består af mange forskellige områder, en af de vigtigste aspekter inden for software-udvikling er "software deployment". Software er ikke færdig før den kan køres, bare fordi man har skrevet noget kode skal man også sikre sig at koden kører ordentlig på de maskiner og servere man har til rådighed. Typisk når man skriver noget kode, så skal den i gennem en længere proces hvor flere udviklere skal reviewe og godkende ens kode. Man benytter bl.a. værktøjet GitHub, som det man kalder for "version-control" eller "versionsstyring" på dansk. GitHub er udviklet for at sikre at den kode man skriver ikke indeholder bugs og fejl, dette gøres gennem "pull-requests", hvor man beder andre udviklere om at se på ens kode før den bliver skubbet til kodebasen. Derefter skal den kode man har skrevet deployes på forskellige servere, hvis man fx skriver backend kode. Man benytter sig at forskellige cloud-providers, som kunne bl.a. være Google Cloud, Amazon Web Services (AWS) eller Microsoft Azure. Udtrykket "cloud-computing" refererer til brugen af virtuelle computerressourcer, som ikke ligger lokalt hos den udvikler eller firma der arbejder på at udvikle software. Man har typisk kæmpe datacentre andre steder i Europa, hvor al den kode man har skrevet bliver uploadet og kørt. Processen for udvikling og deployment kaldes også for "DevOps". "Dev" udtrykket kommer fra ordet "Development", som betyder udvikling, og "Ops" står for "IT Operations", som er de processer vedrørende deployment af selve koden og maintenance.

Datalogiens videnskabsteori

Vi har introduceret jer til den aspekt vedrørende programmering og udvikling inden for datalogiens verden, men datalogi er meget bredere end blot programmering. Programmering er blot en af de praktiske anvendelser af datalogiens teori, men Datalogi har også en teoretisk fundament som kan være interessant at studere for sig selv. Her beskæftiger man bl.a. med spørgsmål om fx Turing-completeness, som er en egenskab af programmeringssprog, tidskompleksitet, som er hvor lang tid det tager at køre en bestemt algoritme, rekursionsteori, kvantecomputerer, Kolmogorovs kompleksitet, og mange flere! Et godt sted at starte på at lære disse emner er gennem Wikipedia, man kan starte med at søge "Computer Science Wikipedia" i Google, og læse hvad man selv finder interessant!

Figur 5.7: Nogle server racks, formentlig i et datacenter

Kompetitiv programmering

Kompetitiv programmering kan beskrives som datalogiens sportsgren. Konceptet går ud på, at man får givet en opgave som man skal løse ved sin datalogiske viden. Ofte skal man få nogle ideer, så man kan restrukturere problemet til noget man ved hvordan man kan løse. Et aspekt af kompetitiv programmering, som gør det svært, er at man ofte skal komme på de bedste løsninger til problemet. Med bedste løsninger, menes der dem der på noget input kan give et svar, hurtigt. Årsagen til dette er at der er en tidsbegræsning på hvor lang tid det må tage koden, at give output ved et givent input. Kompetitiv programmering er ikke bare en god måde at udvide sin datalogiske forståelse, men også et middel til at udvikle sine problemløsningsevner. Der findes mange hjemmesider med problemer, og vi kan varmt anbefale følgende.

- `https://cses.fi/`
- `https://codeforces.com/`
- `https://open.kattis.com/`
- `https://leetcode.com/`
- `https://onlinejudge.org/`

Det anbefales desuden at man starter med problemer som har lav rating og derefter bevæger op. Man kan komme ret langt med betinget udførsel, loops, dictionaries, arrays osv. Derefter har man brug for at lære teori. Det første link har tilknyttet en god bog til at lære basal teori.

Kapitel 6

Geofysik

6.1 Hvad er geofysik?

Geofysik er et fag, som bruger matematik og fysik til at beskrive Jorden. Geofysik dækker over rigtig mange forskellige felter. Et af felterne er undersøgelsen af Jordens indre, hvor man prøver at kortlægge, hvad der sker under Jordens overflade og helt ind til Jordens kerne. Andre felter er bl.a. undersøgelsen af vulkaner og jordskælv, inden for det felt man kalder seismologi, og undersøgelser af Jordens atmosfære.

Et af de allerstørste felter inden for geofysik, er undersøgelsen af klimaforandringer. I nyhederne hører man tit om klimaforandringer, når man hører, at temperaturerne stiger, vandstandende stiger, og mere af indlandsisen smelter. Hvis man lægger mærke til det, så bliver der ofte talt om Grønland, når man taler om klimaforandringer, og det er ikke helt uden grund. Det er nemlig sådan, at når temperaturerne stiger, så påvirker det især de isdækkede områder på Jorden. Derfor er det, som man kalder kryosfæren (en betegnelse for alle isdækkede områder på Jorden), det som I skal undersøge i det her forløb. Kryosfæren er super interessant, fordi så snart der sker ændringer i klimaet, så ser man det tydeligt på, hvor meget af isen, der smelter.

6.2 Gletsjer

Gletsjere er vedvarende is, som bevæger sig under deres egen vægt. Hvad betyder det, at den bevæger sig? Kigger vi f.eks. på en gletsjer på grønland, så flyder is ud mod havet i størrelsesorden meter per år. Det betyder ikke, at gletsjeren behøver at blive mindre, eller flytte sig lige som en mile(sandbakke).

Gletsjere dækker ca. 10 % af jordens overfalde, hvor 99 % af det er på de to store plader Grønland og Anarktis. Hvor den sidste 1% kan findes på bjerggletsjere og andre typer af gletsjer vi finder rundt omkring i verden. Et flot eksempel på en bjerggletsjer er Svartisen i Norge.

6.3 Overflade masse balance

Vi har snakket om at vi kan have en gletscher som ikke bliver større eller mindre, selvom den bevæger sig. De kommer af en ting som hedder masse balance som betyder at der kommer ligeså meget ind som der går ud, Den fulde ligning for det for en gletscher er

$$\dot{b}_s = \dot{a}_s + \dot{a}_a - \dot{m}_s + \dot{a}_r - \dot{s} + \dot{a}_w \tag{6.1}$$

men den ligning bruger vi nærmest aldrig, da vi ofte slår led sammen og derved giver det som

$$\dot{b}_s = \dot{a}_s - \dot{m}_s \tag{6.2}$$

som er at masse balancen er ligmed snefaldet minus smeltningen. Så hvis der falder sne tilsvarende smeltningen så ændre gletscheren ikke størrelse. Dette kan også lede til at

Figur 6.1: Svartis gletscheren i Norge, `https://commons.wikimedia.org/wiki/File:Svartisen_Glacier;_Day_Four_of_Hurtigruten_Coastal_Voyage_North_(21).jpg`

finde gennemsnitshastigheden, hvis systemet er ligevægt, det betyder at masse balancen er nul. Ligevægt er noget vi ofte bruger for at få det til at være lettere ift. hvordan gletscheren bevæger sig for at kunne regne på det. Gennemsnithastigheden er givet ved

$$\overline{u} = \frac{\dot{a}_s x}{H(x)} \tag{6.3}$$

Hvor x er afstanden skillelinjen, og H er højden af gletscher, som ændres langs x, men udtrykket for højden er meget kompliceret. Den anden ting man kan se er at overflade hastigheden kan findes udfra gennemsnitshastigheden, som er

$$u_s = \frac{5}{4}\overline{u} \tag{6.4}$$

Man kan se den ændres over stor dele af grønland se

- **Opgave 6.3.1:**

  1)
  Hvad er masse balancen hvis der smelter $0{,}35\,\mathrm{m\,yr^{-1}}$ og der falder $1\,\mathrm{m\,yr^{-1}}$ i is?

  2) Hvad betyder det, at tallet er positivt?

  3) Hvad skal smeltningen være hvis der falder $1\,\mathrm{m\,yr^{-1}}$ i is, for at den er i ligevægt?

- **Opgave 6.3.2: I**

  opgaverne her skal I aflæse på figuren for at finde noget information i skal bruge.

  1) Hvad er gennemsnitshastigheden for gletscheren ved sne fald svarende til $0{,}085\,\mathrm{m\,yr^{-1}}$, og temperatur på $-10\,^\circ\mathrm{C}$, ved 150 km?

  2) Hvad er overflade hastigheden for gletscheren ved sne fald svarende til $0{,}35\,\mathrm{m\,yr^{-1}}$, og temperatur på $-4{,}5\,^\circ\mathrm{C}$, ved 200 km?

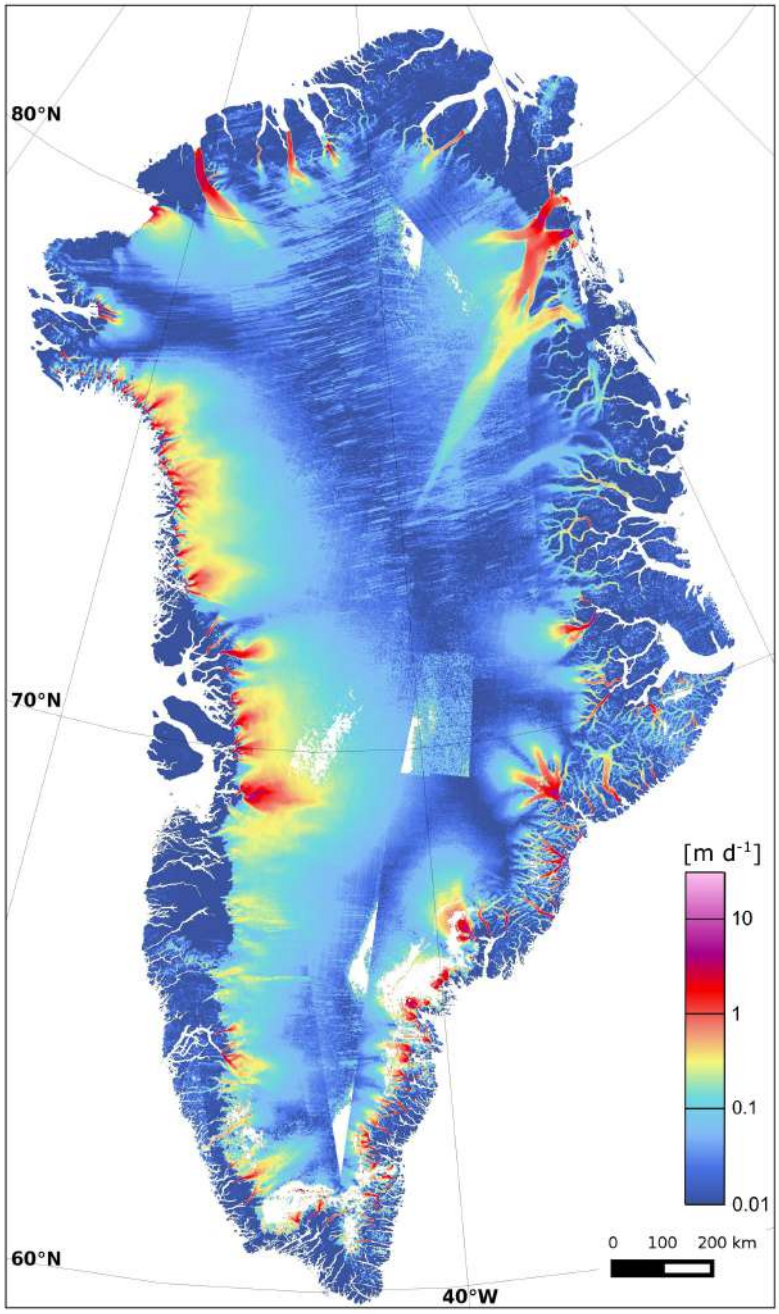
  3) Hvad er sne faldet for den blå linje hvis gennemsnitshastigheden ved 200 km er $28\,\mathrm{m\,yr^{-1}}$?

- **Opgave 6.3.3: N**

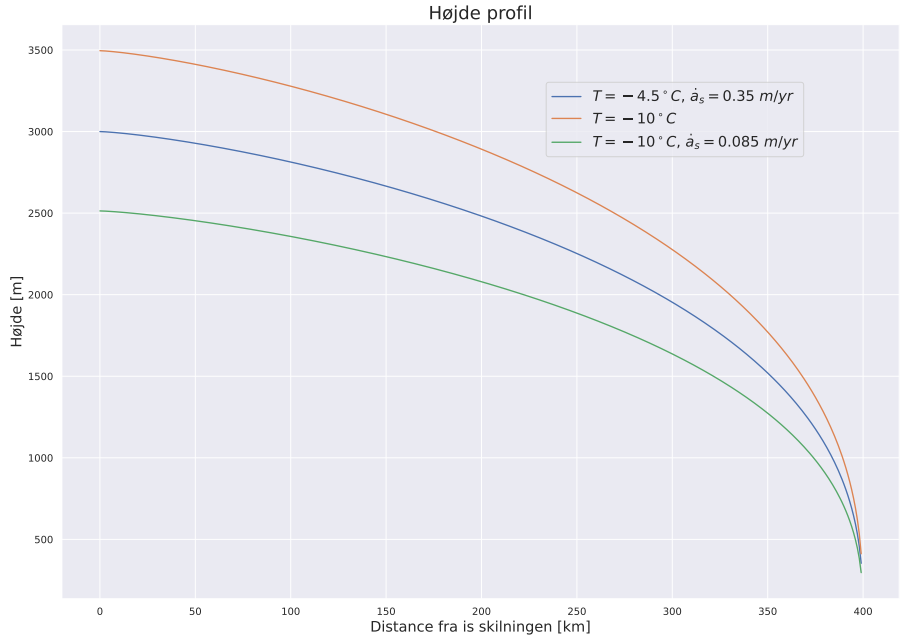
  u prøver vi og se på et eksempel som er mere ligesom virkeligheden, I stedet for at der falder sne over hele området og der forsvinder så det betyder

$$\dot{b}_s = \dot{a}_s x_e - \dot{m}_s(L - x_e) \tag{6.5}$$

Hvor x_e er ligevægtslinjen, så der går lige så meget ind som der går ud og L er længden fra is skilningen.



Figur 6.2: Den viser hastigheden af overfladen på gletscherne på grønland, i enheden meter per dag, `https://www.esa.int/ESA_Multimedia/Images/2015/11/Ice_sheet_in_motion`



1) Hvis gletscheren er 400 km lang, og der falder 0,35 m yr^{-1} hvor langt fra is skilningen er ligevægtslijnen?

6.4 Kryosfæren

Krysosfæren er et begreb som beskriver alle områder på Jorden dækket af is. Der er nogle forskellige elementer i kryosfæren, som det er godt at kunne skelne imellem. En af hovedelementerne er indlandsisen, som er isdækkede områder der har et areal der er større end 50.000 km^2. Det er for eksempel områder som Grønland og Antarktis. Hvis det isdækkede område er mindre end 50.000 km^2, så kalder man det en iskappe.

Hvis man kigger på en mindre skala, så har man at gøre med en gletsjer. En gletsjer er en masse af is, som bevæger sig over land. Gletsjere eksisterer mange steder på Jorden, men man finder dem især ved polerne, Grønland og Antarktis, eller for eksempel også i Nordamerika.

Indtil videre har vi kun talt om den del af kryosfæren der er på land, men der eksisterer faktisk også is på vandet. Det kalder man havis, fordi det er is som er skabt ude i havet.

6.5 Havis

Når man undersøger de områder på Jorden som er dækket af is, så ligger meget af isen faktisk ude i havet. Det er den del af de isdækkede områder som man kalder havis. Havis er anderledes fra gletsjere, fordi den is der bliver dannet bliver skabt ude i havvandet. Det betyder altså, at havisen bliver skabt, når havvandet når sin frysetemperatur som sker ved -1.8 grader. Hvis toppen af havvandet når denne temperatur, begynder havvandet at fryse og samle sig til en klump is. Denne klump af is bliver større og større, desto mere vand under klumpen af isen der når frysepunktet. Det betyder altså, at havisen vokser nedad.

Ligesom sne og is, vokser havis om vinteren, når havvandet bliver nedkølet til frysetemperaturen, og smelter i sommermånederne, når havvandet bliver varmet op. Selvom man har en smelteperiode om sommeren, så er det faktisk ikke alt havis der smelter fuldstændig. Det handler nemlig om, hvor tyk havisen er nået at blive inden man når smelteperioden.

Figur 6.3: Hvilken type havis tror I, at billedet viser?

Typer af havis

Man kan inddele havis i to kategorier: et-årig is og fler-årig is. Som man kan høre på navnet, så inddeler man havisen efter hvor gammelt det er. Hvert år kommer der en kold vinterperiode og en varm sommmerperiode, som hhv. fryser og smelter havisen. Hvis havisen der bliver skabt om vinteren ikke når at vokse sig tyk nok, så smelter alt havisen væk om sommeren. Det betyder altså, at havisen ikke når at overleve sommerperioden, og derfor er mindre end et år gammelt - altså et-årig is. Hvis derimod, at havisen når at blive tyk nok igennem vinterperioden, vil der stadig være noget af havis klumpen tilbage, når man når vinterperioden igen. Derfor kan havisen i den næste vinterperiode nå at vokse sig endnu tykkere, og kunne overleve mere end 1 sommerperiode - altså fler-årig is.

Når temperaturerne stiger om sommeren, vil der smelte mere havis om sommeren, hvilket medfører, at der vil være mere et-årig is end fler-årig is. Det betyder altså, at når der sker klimaforandringer, som får temperaturerne til at stige, så vil der være mindre og mindre havis der kan overleve en smelteperiode om sommeren.

Forskellen på de to istyper er altså, at fler-årig is generelt er tykkere end et-årig is, hvor et-årig is typisk er under 2 meter tykt, og fler-årig is er tykkere end 2 meter.

Ofte kan man kende forskel på et-årig is og fler-årig is ved at kigge på overfladen af isen. Når havisen flyder rundt i havet, er der en sandsynlighed for at to forskellige havis klumper kan støde ind i hinanden. Når det sker, så kan den ene isklump blive ødelagt, hvor den vil ændre form. Her kan der enten blive skabt et isbjerg, hvis havisklumperne sætter sig sammen, eller også kan en havisklump revne pga. sammenstødet. Det gør, at overfladen bliver mere ru. Fordi at fler-årig is flyder rundt i længere tid og er tykkere end et-årig is, så vil den igennem sin levetid have stødt ind i flere klumper is, og vil derfor tit have en overflade der er meget mere ru end et-årig is.

- **Opgave 6.5.1:**

1) Kan I gætte ud fra billedet nedenunder, hvilken type havis det er?

Termodynamisk ligevægt

Nu kan man begynde at tænke, at fler-årig is kan blive uendeligt tykt, og det kan det selvfølgelig ikke. Der er rent faktisk en grænse for, hvor tykt havisen kan vokse til. Den grænse kalder man for punktet ved termodynamisk ligevægt. Termodynamik er et ord

som beskriver varmetransport, hvoraf termo betyder noget med varme, og dynamik beskriver en slags transport. Altså, handler termodynamisk om hvordan noget termisk energi (varme) fra et sted bliver transporteret til et andet sted.

For havis, er punktet med termodynamisk ligevægt det punkt, hvor isen ikke kan vokse sig tykkere. Når havis vokser, så er havet omkring havisen relativt varm i forhold til temperaturen af luften. Her er det altså luftens temperatur, som skal afkøle vandet nedenunder havisklumpen for at havisklumpen kan vokse sig større. Når havisklumpen vokser sig større, så er der længere ned til det vand som skal afkøles for, at havisen kan blive tykkere. Det betyder faktisk, at havisklumpen fungere som et isolerende lag for det vand som er nedenunder den, altså noget som blokerer for, at den termiske energi energien (kulden fra luften) kan transporteres ligeså nemt ned til vandet under havisklumpen. Desto tykkere havisen har vokset sig, desto langsommere vil isen vokse. På det tidspunkt hvor isen har vokset sig så tyk, så det isolerende lag af is er for tykt til at noget kulde kan nå ned til vandet under isklumpen, så har havisen nået punktet for termodynamisk ligevægt. Altså betyder ordet ligevægt, at havisen har nået et punkt, hvor der ikke sker nogle ændringer, og havisen ikke længere vokser.

6.6 Archimedes lov

Archimedes lov beskriver hvordan objekters opdrift er i fluider. Så hvad er en fluid, fluid er en sammensætning mellem væsker og gasser, det betyder at Archimedes lov både kan bruges til at beskrive hvordan is flyder i vand og hvordan balloner flyver op i luften. Archimedes lov siger at opdriften er ligmed massen af den fortrængte fluid ganget med tyndekraften. Så

$$F_{op} = m_{fortrængt} g = \rho_{fluid} V g \tag{6.6}$$

ρ er densiteten af fluiden, og V er volumen af obejktet nedsænket i fluiden. Så f.eks. hvis vi har en badebold med volumen på $0{,}5\,\mathrm{m}^3$, og vi skænker den ned i vand som har en densitet på $1000\,\mathrm{kg\,m^{-3}}$, det betyder vvi kan finde opdriften ved

$$F_{op} = 0{,}5\,\mathrm{m}^3 \cdot 1000\,\mathrm{kg\,m^{-3}} \cdot 10\,\mathrm{m\,s^{-2}} = 5000\,\mathrm{N} \tag{6.7}$$

Det at vi for en opdrifts kraft siger ikke så meget, men hvis vi omskriver den til at vi også bruger tyndekraften, som virker modstat af opdriften.

$$F_{res} = (\rho_{fluid} - \rho_{objekt}) V g \tag{6.8}$$

Hvis den resultere kraft er positiv, så vil objekt flyde opad, mens hvis den er negativ så vil den synke. Så hvis den resluterne kraft er nul, så vil objektet stå stille. Det kan vi bruge til at finde udaf hvor meget der vil være over fluiden, når den ligger stille. Det gøre vi ved at ændre ligning fra før til

$$\rho_{fluid} V g = \rho_{objekt} V g \tag{6.9}$$

Så anser vi objektet for at være en kube, som så giver

$$\rho_{fluid} h^3 g = \rho_{objekt} h^3 g \tag{6.10}$$

Archimedes lov virker kun på den del som er nedsænket i fluiden, så derfor bliver det

$$\rho_{fluid} h_{nedsænket} = \rho_{objekt} h \tag{6.11}$$

så for at finde højden af det som er nedsænket

$$h_{nedsænket} = \frac{\rho_{objekt}}{\rho_{fluid}} h \tag{6.12}$$

Eksempel: Hvis vi gerne vil finde hvor mange procent af en is klump, først ser vi på hvad densiteten af is er, her bruger vi gletscher is, hvilket giver en densitet på $917\,\mathrm{kg\,m^{-3}}$. Så finder jeg procentdelen som er under hvilket er

$$\frac{h_{nedsænket}}{h} = \frac{\rho_{objekt}}{\rho_{fluid}} \tag{6.13}$$

Så sætter jeg tal ind

$$\frac{917\,\mathrm{kg\,m^{-3}}}{1000\,\mathrm{kg\,m^{-3}}} = 0.917 \tag{6.14}$$

Så for at finde hvor meget der er over bruger jeg at det er 1 minus fraktionen også ganget med 100 for at få det i procent.

$$(1 - 0{,}917) \cdot 100 = 8{,}3\,\% \tag{6.15}$$

- **Opgave 6.6.1:**

1) Hvis vi har en sten med densitet på $2000\,\mathrm{kg\,m^{-3}}$ i vand, hvad er den resulterende kraft?

2) Hvor stor en procent del af sten fra opgaven før vil være over vandet?

- **Opgave 6.6.2:**

1) Hvis vi har en helium med densitet på $0{,}1\,\mathrm{kg\,m^{-3}}$ i saltvand med en densitet på $1025\,\mathrm{kg\,m^{-3}}$, hvad er den opdrifts kraft?

2) Hvad vi den resulterende kraft blive? Kraft er det samme som masse gange acceleration. Så

$$F = ma \tag{6.16}$$

3) Hvor meget vil ballonen accelerere?

- **Opgave 6.6.3:**

1) Hvis vi har gletscher is med en densitet på $917\,\mathrm{kg\,m^{-3}}$ som er 50 m høj, nedsænket i saltvand. Hvor meget vil vi se over overfladen? Vi ser bort fra luftens densitet, som er $1{,}225\,\mathrm{kg\,m^{-3}}$, hvis vi ikke gør det bliver ligningen for nedsænket højde

$$h_{nedsænket} = \frac{h(\rho_{luft} - \rho_{objekt})}{(\rho_{luft} - \rho_{fluid})} \tag{6.17}$$

2) Hvis vi har gletscher is med en densitet på $917\,\mathrm{kg\,m^{-3}}$ som er 50 m høj, nedsænket i saltvand. Hvor meget vil vi se over overfladen hvis vi tager højde for luftens densitet?

3) Ville I tror vi kan måle effekten i den virkelige verden?

- **Opgave 6.6.4:**

1)
Hvis vi har et objekt med en højde på 10 m og der er 1 m over vandoverfladen, Hvad er densiteten af objektet?

Hvorfor er havis interresant?

Mængden af havis har ændret sig rigtig meget over årene. Når der kommer nogle lange sommermåneder, hvor det er virkelig varmt, så påvirker det hvor meget havis der er oppe ved Arktis området, det som man kalder havis koncentration. Det betyder altså, at klimaforandringer virkelig bliver tydelige at se på mængden af havis. Herudover, så kan man se over de sidste mange år, at der kommer mindre og mindre fler-årig is og mere et-årig is. Det skyldes primært, at der er generelle temperaturstigninger, som gør at havisen simpelthen ikke når at fryse nok til at holde en hel smeltesæson, og det hele bare vil smelte væk.

Det betyder altså, at vi har en ret stor kobling mellem klimaforandringer og mængden af havis, og det har bl.a. også noget at gøre med, at når der er mere havis, så vil albedoen af havisen stige. Det term er rigtigt vigtigt, og det skal I lære hvad er nu.

6.7 Albedo

Albedo er et fænomen som man ofte undersøger især, når man arbejder med is. Når solen skinner på Jorden, så kan man se det som en lysstråle som bliver sendt ud fra Solen, som rammer Jorden og på Jordoverfladen bliver reflekteret tilbage til rummet. Når strålingen fra Sollyset rammer overfladen, bliver den ikke reflekteret fuldstændigt, men noget af sollyset vil blive absorberet af jordoverfladen. Forholdet mellem hvor meget af sollyset der bliver udsendt i forhold til hvor meget der bliver reflekteret af overfladen, kaldet man en overflades albedo. Det kan man skrive op med en ligning:

$$\text{Albedo} = \frac{\text{Reflekteret sollys}}{\text{Udsendt sollys}} \tag{6.18}$$

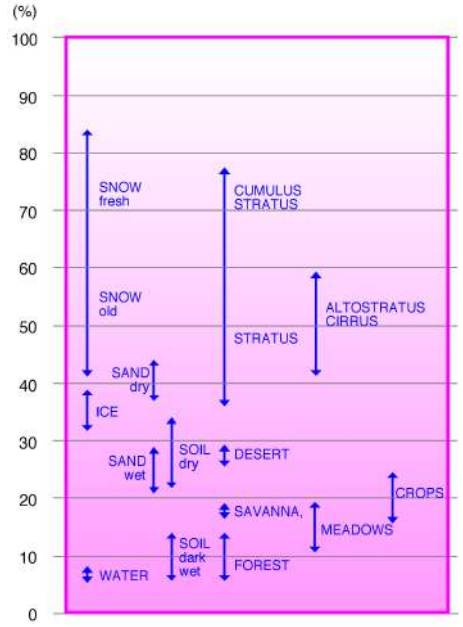
Ofte skriver man albedo op som en procentsats. Det betyder altså, at hvis alt sollyset bliver reflekteret af jordoverfladen, så siger man at den overflade har en albedo på 100 %, og omvendt hvis alt sollyset bliver absorberet har overfladen en albedo på 0 %.

Det som er især interessant er, at ikke alle overflader har den samme albedo. F.eks. har sne som lige har lagt sig en rigtig høj albedo, altså hvor næsten alt sollyset bliver reflekteret, ift. åbent vand som har en meget lavere albedo, hvor meget af sollyset bliver absorberet af overfladen. På figuren nedenfor kan I se albedo værdierne for forskellige materialer: På figur 6.4 kan man se, at frisk sne, altså sne som lige har lagt sig, har den højeste albedo, og vand har den laveste albedo. Herudover kan man se albedo værdien for nogle forskellige typer skyer (cumulus stratus, altostratus cirrus), som har en meget høj albedo værdi. Læg især mærke til at is har en lavere albedo værdi end sne. Det betyder altså, at isdækkedeområder med sne oven på sig vil reflektere mere af solens lys, end is uden snedække. Det kan vi faktisk trække tilbage til, hvordan havisen vokser. Hvis der ligger et lag oven på isen, så vil sneen fungere som et isolerende lag, så det forhindrer luften i at kunne afkøle vandet under havisen for at havisen kan vokse. Sneen fungerer netop som et isolerende lag, fordi den har en højere albedo end den underliggende havis klump, og den derfor reflekterer mere af sollyset og forhindrer en udveksling af varme mellem luften og vandet under havisklumpen.

- **Opgave 6.7.1:**

1) Prøv at forestille jer et scenarie, hvor der ligger sne ovenpå et stykke havis. Hvad sker der når sneen smelter? Hvordan vil albedoen ændre sig?

Så kan man spørge sig selv, hvorfor det er vigtigt at vide hvor meget af sollyset der bliver absorberet. Det er faktisk virkelig vigtigt at forstå, fordi desto mere af sollyset der bliver absorberet, desto varmere bliver overfladen. Det er noget man for eksempel kan se om sommerdag, hvor der har været bagende sol hele dagen. Hvis man rører ved



Figur 6.4: Albedo værdier for forskellige materialer.

en sort bil ift. en hvid bil, så vil den sorte bil altid vil være varmere end den hvide bil. Det sker netop, fordi den sorte bil har en lavere albedo - den reflekterer altså mindre af solens lys.

Opgaver med albedo

Nu kan I prøve at udregne albedo værdien for en station i Grønland. Det er en PROMICE vejrstation som bl.a. måler albedo. [8] Stationen hedder Promice ligger i nordøst Grønland og hedder Kronprins Prins Land (KPL). Denne station ligger ved en gletsjer, hvor den tager to målinger. En måling der bliver taget på toppen af gletsjeren, og en anden måling nede ved bunden af gletsjernen nede mod fronten. Nu skal I prøve at regne albedoen for målingen taget øverst på gletsjeren og nederst på gletsjeren:

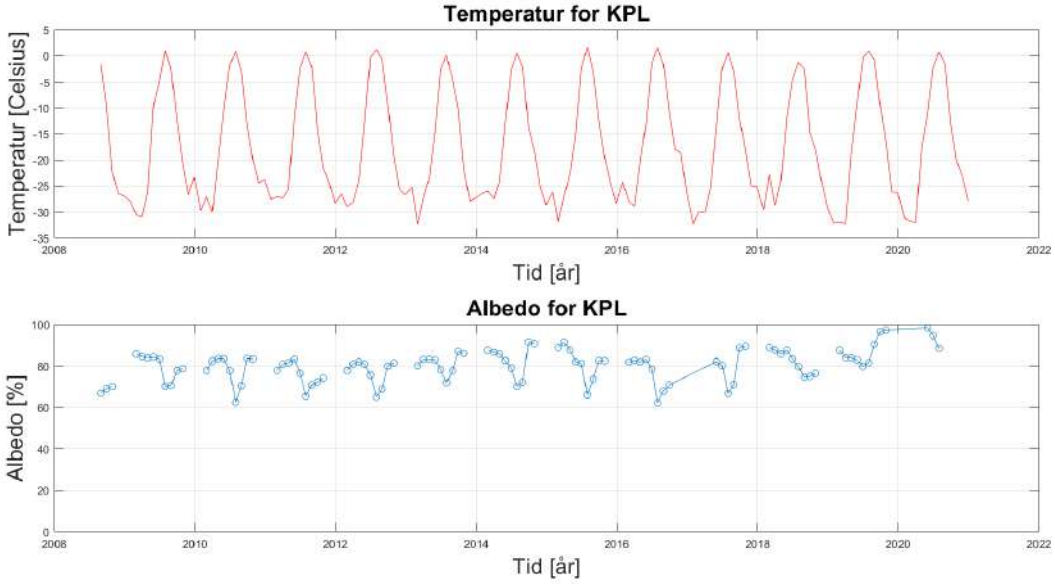
- **Opgave 6.7.2:**

1) På toppen af KPL stationen har I givet en reflekteret stråling på 127.8 $\frac{W}{m^2}$ og en udsendt stråling på 191.4 $\frac{W}{m^2}$. Udregn albedoen.

2) På bunden af KPL stationen har I givet en reflekteret stråling på 66.2 $\frac{W}{m^2}$ og en udsendt stråling på 112.5 $\frac{W}{m^2}$. Udregn albedoen.

3) Hvilken af af albedoerne er højest? Hvis I tænker på, at toppen af gletsjeren har mere snefald en den nederste del af gletsjeren, kan I så regne ud, hvorfor der er den her forskel i albedo for de to målinger?

Når I har regnet på denne station, kan I se noget data nedenunder fra stationen over en længere tidsperiode fra 2008 til 2020. Her er der blevet plottet, hvordan temperaturen ændrer sig samtidigt med et plot af hvordan albedoen ændrer sig i løbet af denne tidsperiode.



Figur 6.5: Plot af data fra KPL stationen. Øverst ser i temperaturen plottet og nederst ser I albedoen plottet.

Nu skal I prøve at analysere det her data:

- **Opgave 6.7.3:**

1) Kan I se en sammenhæng mellem temperaturen og albedoen? Hvad sker der generelt når temperaturen er høj med albedoen?

2) I albedoen kan I se, at der er nogle huller i dataen. Hvorfor tror I de her huller er der? Læg mærke til at de her huller især sker i vinterperioderne.

6.8 Hvad er lys?

Når man skal ind at kigge på solens lys, så er det vigtigt at forstå, hvad lys i virkeligheden er. Det er faktisk sådan, så det lys som kommer fra Solen kun er en lille del af det som hedder det elektromagnetiske spektrum. Det elektromagnetiske spektrum er en måde at inddele lys i forskellige kategorier afhængigt af deres bølgelængde og deres frekvens, hvor de har følgende sammenhæng:

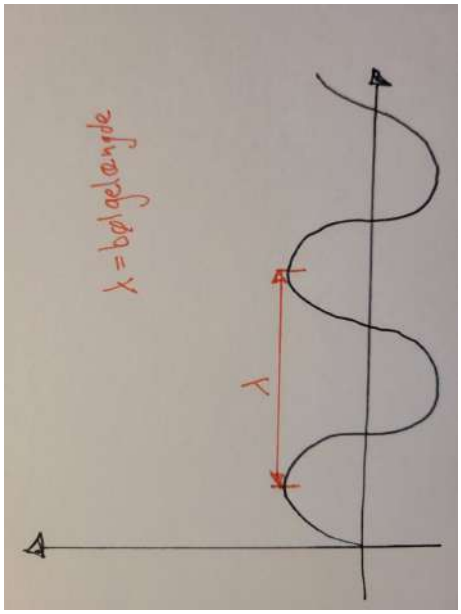
$$c = \lambda \cdot f \tag{6.19}$$

hvor c er lysets hastighed, λ er bølgelængden og f er frekvensen.

Og hvad er det så lige at bølgelængde og frekvens er? Hvis vi kun fokuserer på bølger der gentager sig selv, og derfor er periodiske, så kan man, når man kigger på en bølge, se at den på et tidspunkt vil begynde at gentage, hvordan den bevæger sig. En nem måde at finde bølgelængden på er at kigge på afstanden fra en bølgetop til en bølgetop, se billede 6.6. Når man støder ind i en bølgetop igen, så vil den tid der er gået mellem bølgetop til bølgetop svare til en periode. Bølgelængde måles som en hvilken som helst anden længde, og måles derfor f.eks. i meter.

Frekvens hænger faktisk sammen med perioden. Det er en parameter der siger, hvor mange bølgetoppe man når at støde ind i, i løbet af et sekund. Den måles i enheden Hertz, som svarer til $\frac{1}{\text{sekund}}$, og man kan regne den sådan her:

$$f = \frac{1}{T} \tag{6.20}$$



Figur 6.6: Bølgelængde for lys.

hvor T er perioden.

Det som I kan se fra ligning 6.8 er, at siden venstresiden (lysets hastighed) er en konstant, så vil der være et forhold mellem bølgelængden og frekvensen. Det er nemlig sådan, at for at venstresiden er konstant, så hvis vi har en større bølgelængde, så bliver frekvensen mindre og omvendt.

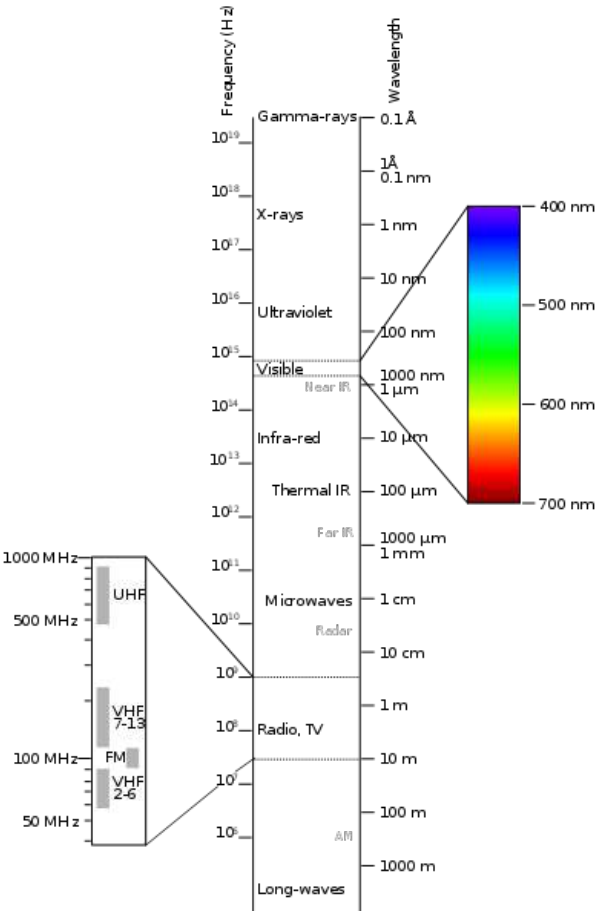
Nu hvor vi har en lidt bedre forståelse for, hvilke elementer der skal til for at beskrive en lys bølge, kan vi gå lidt mere i dybden med det elektromagnetiske spektrum. Det elektromagnetiske spektrum kan man se nedenunder på billedet: På spektrummet ovenfor, så ligger det meste af det lys som solen udsender i det man kalder det kortbølgede spektrum, og sender faktisk lys ud i både den synlige, infrarøde og den ultra violette (UV) del af det elektromagnetiske spektrum. På billedet kan man se opdelingen af det synlige lys i de forskellige farver, hvor den del af det synligt lys med de korteste bølgelængder har en blå farve og de lyset med de længste bølgelængder har en rød farve.

Absorbering af sollys i atmosfæren

Der sker noget interessant med lyset fra solen, når det bevæger sig igennem atmosfæren. Ikke alt solens lys når ned til jordoverfladen, og det sker netop fordi der er en masse ting ude i atmosfæren, som kan forhindre sollyset i at nå ned til jordoverfladen. Atmosfæren består af en masse forskellige ting. Den indeholder bl.a. vanddamp, kuldioxid og ozon, som svarer til 3 oxygen atomer bundet sammen. Det betyder derfor også, at desto flere skyer der er på himlen, som i høj grad består af vanddamp, desto mere af sollyset vil blive reflekteret og ikke nå ned til jordoverfladen.

Her er det især den del af sollyset som er i det ultraviolette spektrum som bliver reflekteret. Og det er faktisk en rigtig god ting! Det er det nemlig, fordi de stråler der ligger i den yderste del af UV spektrummet, hen mod X-ray stråling, er de stråler som kan være mest skadelige for mennesker. Det er især ozonlaget der er godt til at beskytte mod den slags stråling.

Ligesom ozonlaget er rigtig godt til at absorbere den ultra-violette stråling fra solen, så kan man faktisk finde ud af, hvordan forskellige molekyler absorbere de forskellige dele af sollyset. Nu skal vi til at bruge det I lige har lært om bølgelængder.

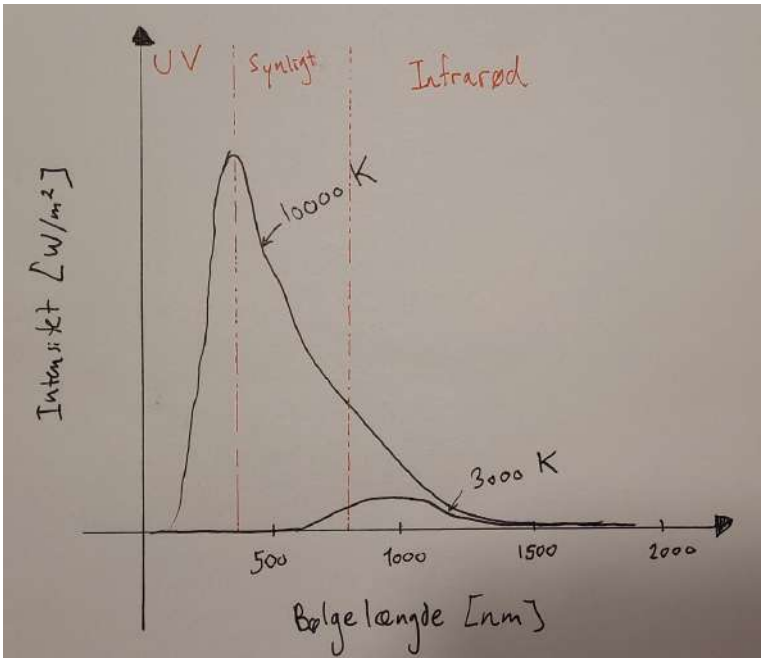


Figur 6.7: Det elektromagnetiske spektrum med frekvens i venstre søjle og bølgelængde i højre søjle.

Sort legemes stråling

Når man skal kigge på, hvordan forskellige molekyler absorbere sollys, så skal man have en forståelse for, hvordan et element absorbere lys.

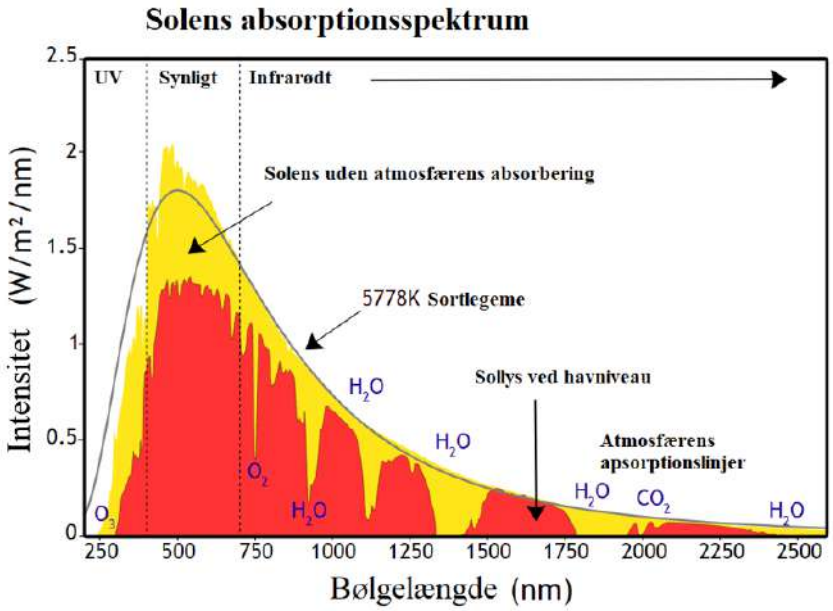
Her skal man kende til begrebet der hedder sortlegeme. Man antager, at et sortlegeme er et legeme der absorberer alt sollys. Det er faktisk derfor det hedder et sort legeme, fordi alt sollyset bliver absorberet, og intet lys reflekteres, og det derfor bliver helt sort. Det er af samme grund, at man kalder et sort hul for et sort hul, fordi det absorbere alt sollys. Mange kendte fysikere, bl.a. Max Planck, begyndte i slutningen af 1800 tallet og begyndelsen af 1900 tallet at undersøge den stråling som de sorte legemer udsendte ved forskellige bølgelængder. Her fandt han ud af, at der faktisk er en direkte kobling mellem intensiteten af den stråling det sorte legeme udsender (målt i watt per areal) og den temperatur den er ved. Den sammenhæng kaldes Plancks lov, og nogle eksempler på intensitetskurver ved forskellige temperaturer kan I se nedenfor:



Figur 6.8: Illustration af to forskellige Planck kurver, ved 3000 K og 10000 K.

Her kan I se på billedet ovenfor se, at desto lavere temperaturen er på det sorte legeme, desto længere over mod højre vil toppen på kurven være. Altså, vil det være ved en højere bølgelængde, at vi har den største mængde udstråling fra det sorte legeme.
Nu hvor vi kender det generelt for sorte legemer, kan vi begynde at kigge på den samme slags kurve for sollys.

Overflade temperaturen på Solen er omkring 5778 K. Derfor, når man kigger på spektrummet af Solen, så kigger man en kurve ind for et sortlegeme ved omkring denne temperatur som en reference kurve for hvis Solen absorberede alt lys. Det gør Solen dog ikke, og vil derfor opføre sig lidt anderledes end et sortlegeme. Nedenunder kan I se et billede hvor man sammenligner de to kurver:



Figur 6.9: En graf for sortlegeme strålingen (sort graf) sammenlignet med solens stråling (gul og røde grafer).

Det første man kan se på kurven er, at den gule del af kurven svarer til stråling der sker ved toppen af atmosfæren, og den røde del af kurven er den stråling der bliver udsendt ved havniveau. Alle dyk på kurven vil svare til at der er et element i atmosfæren der har absorberet sollyset. Her kan man se, hvor nogle af de forskellige elementer i atmosfæren absorbere lys aller mest. Her ser vi bl.a., som nævnt tidligere, at ozon (O_3) absorberer især det ultraviolette lys. Herudover kan man se, at den infrarøde del af solens stråling primært bliver absorberet af vanddamp, H_2O, i atmosfæren og kuldioxid, CO_2. Siden der ikke er nogle dyk omkring toppen af kurven i det synlige lys, så betyder det, at der ikke er noget i atmosfæren som absorberer lys der udsendes ved disse bølgelængder. Det betyder, at stråling der udsendes ved disse bølgelængder, når hele vejen ned til jordoverfladen.

6.9 Forsøg med albedo

Nu kommer vi til at skulle lave et forsøg til at måle albedo. Et pyranometer er et instrument der kan måle det som man kalder kortbølget stråling, som er præcis det som kommer fra Solen af, altså stråling i det ultraviollette område til det infrarøde område som I så i det elektromagnetiske spektrum.

Pyranometeret er en sensor, som måler lysintensiteten per areal, som er i enheden Watt per kvadratmeter. Det som I skal begynde med er at vende sensoren op mod solen. Herefter skal I måle i et par sekunder og herefter vende sensoren om, så sensoren har fronten ned mod jorden. På den måde har I nu 2 målinger: En som måler det samlede udsendte lys fra solen, som ikke er blevet absorberet af atmosfæren og en som måler hvor meget af den stråling som bliver reflekteret af jordoverfladen. Når I har lavet disse målinger, skal i bruge ligning 6.7 til at udregne albedo værdien.

Når I finder en albedo værdi ud fra disse målinger, kan I udregne hvor meget der bliver reflekteret af jordoverfladen.

For at lave der her forsøg skal I gøre følgende:

- Installer programmet LoggerPro Lite på linket https://www.vernier.com/downloads/logger-lite-updates/
- Tilsæt pyranometer i Labquest i din pc

- Kallibrer instrumentet ved at trykke på "Forsøg" og så på "Kalibrer" og sæt nulpunktet til 0
- Tryk på "Forsøg" og så på "Data opsamling" og sæt varighed på 10 sekunder. Det betyder at jeres måler giver jer en måling i programmet hvert tiende sekund
- Tryk på "Forsøg" og så på "Tag data"
- Vend først pyranometret opad og herefter nedad.
- Tag en ny måling, hvor I måler over en hvid plade
- Tag en ny måling, hvor I måler over en sort plade

Nu hvor I har fået taget jeres målinger, kan I prøve at besvare følgende spørgsmål:

- **Opgave 6.9.1:**

1) Hvad er albedoen af asfalten? Hvordan kan I sammenligne den værdi I får med albedo værdierne fra figur 6.4.

2) Kan I forklare, hvorfor der er en forskel på albedo værdien over den sort flade og den hvide flade?

6.10 Feed-back mekanismer

Nu har I både fået lidt forståelse for, hvad albedo er og har prøvet at måle det, kan vi nu sætte albedo ind i et lidt større perspektiv. Tit, når man kigger på albedo eller andre fænomener i geofysik, så for at forstå hvordan det påvirker klimaet, undersøger man det man kalder feed-back mekanismer. Feedback mekanismer er en måde at se, hvad konsekvenserne er af at noget sker. Vi kan kigge på et eksempel med albedo.

Når temperaturen stiger, så vil mere is smelte. Når mere is smelter, så vil der komme mere vand. Vand har en lavere albedo en is, hvilket derfor betyder, at mere af solens lys vil blive absorberet af overfladen. Det betyder derfor, at temperaturen vil stige endnu mere, fordi jorden nu lagrer mere varme i sig, fordi albedoen falder. Denne proces kan man skrive op på den her måde:

$$\text{Temperatur stiger} \Rightarrow \text{Is smelter} \Rightarrow \text{Albedo falder} \Rightarrow \text{Temperaturen stiger} \tag{6.21}$$

Det betyder altså, at fordi temperaturen startede med at stige, så medførte det faktisk i sidste ende, at temperaturen steg endnu mere. Den slags mekanisme kalder man en positiv feedback mekanisme. Altså, en positiv feedback mekanisme er, når en effekt, for eksempel at temperaturen stiger, bliver forstærket. Denne feedback mekanisme viser netop hvorfor albedo har så stor en betydning, fordi det kan forstærke klimaforandringer rigtig meget.

Udover positive feedback mekanismer har vi også negative feedback mekanismer. Det er netop det modsatte, fordi det gør at en effekt bliver formindsket. Vi tager endnu et eksempel.
Temperaturen stiger. Når temperaturen stiger, så vil der blive skabt mere vanddamp, og det vil lede til at der kommer flere skyer. De skyer der så er i atmosfæren kan faktisk være med til at reflektere mere af solens stråling ud i atmosfæren. Det kunne derfor lede til, at temperaturen derefter faldt, fordi ikke ligeså meget af solens stråling ville nå ned til jordoverfladen og blive absorberet af jordoverfladen. Denne proces kan skrives sådan her op:

$$\text{Temperatur stiger} \Rightarrow \text{Mere vanddamp (flere skyer)} \Rightarrow \text{Albedo stiger} \Rightarrow \text{Temperaturen falder} \tag{6.22}$$

Nu kommer nogle opgaver, hvor I skal skrive feedback mekanismen færdig. Herefter, skal I vurdere om det er en positiv eller negativ feedback mekanisme:

- **Opgave 6.10.1:**

  1) Temperaturen stiger ⇒ Sneen på gletsjeren smelter ⇒ Albedoen ?? ⇒ Temperaturen ??
  Prøv at tænk på, hvad der sker med albedoen når sneen smelter og udfyld de første spørgsmåltegn.

  2) Hvad er konsekvensen af, at albedoen ændrer sig ift. temperatur? Udfyld de sidste spørgsmåltegn.

  3) Er det en positiv eller negativ feedback mekanisme?

- **Opgave 6.10.2:**

  1) Kroppens temperatur stiger ⇒ Man begynder at svede ⇒ Kroppens temperatur ??
  Prøv at tænke på hvorfor man begynder at svede. Er det for at nedkøle eller opvarme kroppen?

  2) Hvad er konsekvensen af, at du begynder at svede ift. kroppens temperatur?

  3) Er det en positiv eller negativ feedback mekanisme?

Bibliografi

[1] Allan Baktoft. *Matematik i virkeligheden Bind 2*. 3. udg. Forlaget Natskyggen, 2017. ISBN: 978-87-92857-15-6.

[2] Jesper Lützen. *Diskrete Matematiske Metoder*. 2. udg. Københavns Universitet, 2019.

[3] Wikimedia. *Image-Koenigsberg - Wikimedia*. 2016. URL: `https://commons.wikimedia.org/wiki/File:Image-Koenigsberg,_Map_by_Merian-Erben_1652.jpg` (bes. 28.12.2019).

[4] Douglas B. West. *Introduction to Graph Theory*. 2. udg. Prentice Hall, Inc., 2001. ISBN: 0-13-014400-2.

[5] J. J. O'Connor og E. F. Robertson. *Dénes König (1884 - 1944) - Biography - MacTutor History of Mathematics*. 2014. URL: `https://mathshistory.st-andrews.ac.uk/Biographies/Konig_Denes/`.

[6] Wiki. 2022. URL: `https://en.wikipedia.org/wiki/Electronegativity`.

[7] Charles Molner og Jane Gair. *Concepts of Biology - First Canadian Edition*. BC-campus. 2015.

[8] R.S. Fausto og D. van As. *Programme for monitoring of the Greenland ice sheet (PROMICE): Automatic weather station data. Version: v03, Dataset published via Geological Survey of Denmark and Greenland*. 2019. URL: `https://doi.org/10.22008/promice/data/aws`.

Velkommen til Sukkertoppen Gymnasium

Sukkertoppen Gymnasium er et stort, veldrevet gymnasium med fokus på naturvidenskab, teknologi og design.
Vores nøglebegreber er høj faglighed, stor Rummelighed og et aktivt og alsidigt studiemiljø med en levende kultur både fagligt og socialt.

Du arbejder med både teori og praksis og får selvfølgelig mulighed for at slippe din indre forskerspire eller opfinder løs i vores topmoderne laboratorier og værksteder.

Hvad enten du drømmer om at deltage i kemi-olympiade eller arrangere weekendlange LAN-parties, er Sukkertoppen stedet for dig.

På Sukkertoppen Gymnasium bliver du en del af et stærkt fagligt miljø, hvor du som elev udfordres til at blive så dygtig som muligt. Det kommer blandt andet til udtryk, når vi hvert år har elever med til DM i teknologi, DM i science, biologi OL, kemi OL, datalogi OL, Georg Mohr og meget mere.

Vores 1100 elever er ambitiøse, målrettede og trives i et godt studiemiljø, hvor der er klare retningslinjer og et godt sammenhold. Vi har et aktivt elevråd og en god dialog mellem eleverne og ledelsen. Det er med til at skabe den gode stemning, du vil opleve på Sukkertoppen.

Det sociale liv er en vigtig del af gymnasiet.
På Sukkertoppen har vi et godt socialt miljø med fester, brætspilscafeer, filmklub, idrætsdage, musikarrangementer, fredagscafeer og meget andet.

Sukkertoppen Gymnasium er stolt af at huse UNF Naturfagsweekend 2021

Dansk Ungdoms Fællesråd - DUF

DUFs lokalforeningspulje støtter de lokale foreningers arbejde for børn og unge i Danmark og Sydslesvig. Puljen kan søges af lokalforeninger og lokale grupper i DUFs medlemsorganisationer og observatørorganisationer.

DUF

DANSK
UNGDOMS
FÆLLESRÅD

Grundstoffernes Periodiske System