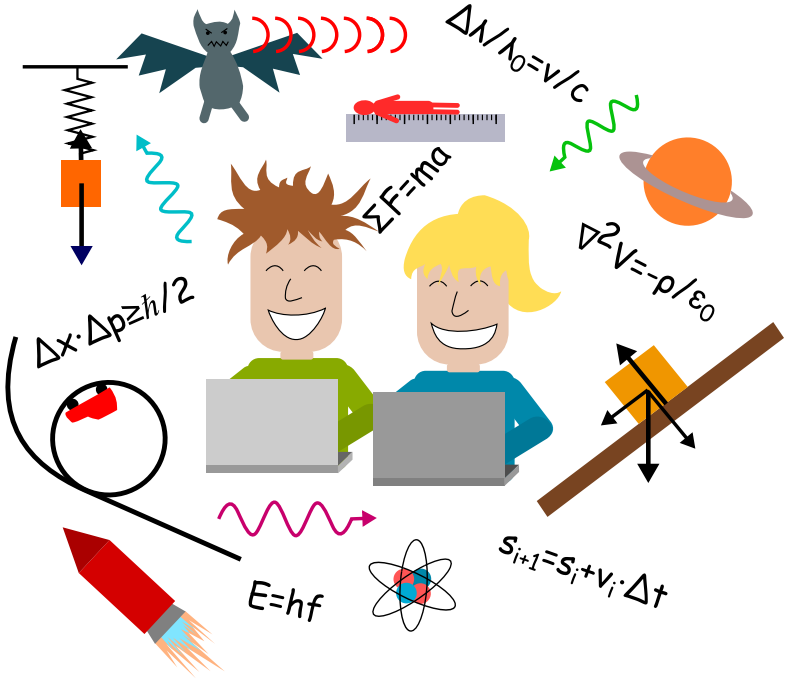
Masteroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for fysikk

Sigurd Torp Nordby

Programmering og algoritmisk tenkning i fysikkundervisning

Masteroppgave i Lektorutdanning i realfag
Veileder: Berit Bungum
Mai 2019



Illustrasjon: Sigurd Torp Nordby

NTNU
Kunnskap for en bedre verden

Sigurd Torp Nordby

Programmering og algoritmisk tenkning i fysikkundervisning

Masteroppgave i Lektorutdanning i realfag
Veileder: Berit Bungum
Mai 2019

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for fysikk

NTNU
Kunnskap for en bedre verden

SAMMENDRAG

Som et ledd i den norske regjeringens strategi for digitalisering av norsk skole, er det bestemt at programmering og algoritmisk tenkning («computational thinking») skal innføres i skolens matematikk- og naturfag. Det har dermed oppstått et behov for forskning om hvordan programmering bør integreres som et gjennomgående verktøy i undervisningen. Denne oppgaven har som formål å bidra til en god, gjennomtenkt innføring av programmering og algoritmisk tenkning i skolens fysikkfag på videregående nivå.

Forskningsarbeidet har hovedsakelig en kvalitativ tilnærming, og arbeidet som helhet har elementer av designbasert forskning. Problemstillingen undersøkes gjennom en litteraturstudie, en intervjuundersøkelse og utvikling og utprøving av undervisningsopplegg. I litteraturstudien gjennomgås allerede eksisterende forskning og erfaringer med programmering og algoritmisk tenkning i fysikk. I intervjuundersøkelsen undersøkes hvordan tre norske fysikklærere forstår begrepet algoritmisk tenkning og hvilke muligheter og utfordringer de ser for innføring av programmering som verktøy i fysikkfaget. Basert på dette er det utviklet tre undervisningsopplegg som er prøvd ut i tre forskjellige fysikklasser ved en videregående skole i Norge. Erfaringer om hvordan lærere og elever opplever undervisningen, er presentert basert på data fra observasjon og spørreundersøkelser.

Resultatene peker på at programmering og algoritmisk tenkning har en naturlig plass i det tjueførste århundrets fysikkundervisning. Å kunne programmere og forstå hvordan algoritmer fungerer er viktig for både arbeidsliv og allmenndannelse. Med programmering som verktøy kan elevene dessuten løse andre typer fysikkproblemer, som ellers ville vært for vanskelige. Det gir mulighet for større endringer i fysikkfagets form og innhold. Lærerne er i all hovedsak positive til programmering, og de ser muligheter for hvordan det kan inkluderes i fysikkundervisningen. Utprøvingen viser at elever uten programmeringserfaring fint klarer å arbeide med nokså avansert programmering i en fysikksammenheng. Lærerne er imidlertid noe bekymret for at programmering kan ta for mye tid, og at fysikkfaget vil bli enda vanskeligere. Undersøkelsen viser også at en del elever opplever undervisningen som svært utfordrende, og det er en fare for kognitiv overbelastning. Studien viser også at den norske oversettelsen av «computational thinking» kan være uheldig, fordi begrepets innhold kan misforstås av lærerne.

ABSTRACT

As part of the Norwegian government's strategy for digitalization of the primary and secondary education in Norway, it has been decided that programming and computational thinking are going to be introduced through the mathematics and science courses. For this reason, there is a need for research on how programming should be integrated as a general tool in the education. The purpose of this thesis is to contribute to a good, thoroughly considered introduction of programming and computational thinking in the physics courses at upper secondary level.

The research has a mainly qualitative approach, and the work as a whole has elements of design-based research (DBR). The thesis statement is examined through a literature study, an interview study and development and testing of physics lessons with programming. The literature study reviews already existing research on programming and computational thinking in physics. The interview study examines how three Norwegian physics teachers understand the Norwegian term for "computational thinking" (which is literally translated "algorithmic thinking"), and what possibilities and challenges they see for introducing programming as a tool in the physics courses. Based on this, three lessons have been designed and tried out in three different physics classes at an upper secondary school in Norway. How teachers and students have experienced the lessons, is presented based on data from observations and questionnaires.

The results indicate that programming and computational thinking has a natural place in physics education of the 21st century. To programme and understand how algorithms work is important for both working life and general education. Moreover, with programming as a tool, students can solve a different kind of physics problems, that would otherwise be too difficult. This allows bigger changes in the curriculum. The teachers are largely positive about programming, and they see potential for how it could be included in the physics lessons. The try-out shows that students without former programming experience manage to do quite advanced programming in a physics context. The teachers are however somewhat concerned that programming may take up too much time, and that the courses will be even more difficult. The try-out also shows that some students experience the teaching as very challenging, and there is a chance for cognitive overload. The study further shows that the Norwegian translation of "computational thinking" might be unfortunate, as the concept could be misunderstood.

FORORD

Med denne masteroppgaven fullfører jeg nå femårig lektorutdanning i fysikk og matematikk ved NTNU. For ganske nøyaktig fem år siden leverte jeg fra meg eksamensbesvarelsene mine i matematikk og fysikk i idrettshallen ved Elverum videregående skole. Det var antakelig på det tidspunktet i livet at jeg trodde jeg hadde lært det meste som var å lære. Siden har det bare gått nedover. Etter fem år som kombinert fysikk-, matematikk- og pedagogikk-student, har det blitt klart hvor lite jeg egentlig vet. For hvert emne jeg har tatt, og for hver bok jeg har lest, har jeg innsett at det er ti bøker til jeg gjerne skulle hatt tid til å lese. Slik har også arbeidet med litteraturstudien i denne masteroppgaven vært. Heldigvis er det noen som har satt tidsbestemte frister, og selv om Einstein har tuklet mye med vår forståelse av tid, er det en ting han ikke fikk gjort noe med: tiden går alltid fremover.

Jeg vil rette en stor takk til veilederen min Berit Bungum, som tilbød meg denne spennende oppgaven og har holdt meg gående fra første stund, og som ikke minst har gitt uvurderlig veiledning, råd og tilbakemeldinger. Takk også til Jon Andreas Støvneng, som har stilt opp som biveileder og bidratt med fysikkfaglige råd og erfaring på veien.

Videre kan jeg ikke få takket lærerne og elevene som har stilt opp som intervjupersoner og forsøkskaniner, nok. Takk for deres tid og gode tilbakemeldinger, og ikke minst gode humør og engasjement, som har gjort dette til et svært inspirerende arbeid.

Takk også til alle medstudentene på lektorprogrammet som har bidratt til en flott tid som student i Trondheim, og spesielt takk til den lille fysikk-gjengen som holdt ut helt til slutt. Tusen takk til familien som har støttet meg hele veien, og som aldri har vært lenger enn en videosamtale unna. Jeg er veldig glad i dere alle sammen.

Med dette har tiden kommet for å sette et siste punktum som lektorstudent. Jeg vil hilse til alle de fantastiske lærerne der ute, som hver dag gjør en utrolig viktig jobb. Jeg håper denne oppgaven kan gi et nyttig bidrag til fagfornyelsen, og jeg gleder meg til snart å få være med dere på å forme fremtidens skole.

Trondheim, mai 2015

Sigurd Torp Nordby

INNHOLD

1 INTRODUKSJON

1.1 Bakgrunn

Den teknologiske utviklingen etter tusenårsskiftet har gått raskt. Regnekraft som tidligere krevde datamaskiner på størrelse med et helt rom, bærer de fleste i dag alltid med seg i bukselommen. Store deler av hverdagen foregår digitalt, og banktjenester, billetter, bilder og musikk får vi tilgang til gjennom små programmer på datamaskiner og telefoner. Avisene leser vi stort sett på nett, og vennene våre holder vi kontakt med digitalt gjennom sosiale medier. Algoritmer styrer hvilke annonser vi blir eksponert for. Mobiltelefonen forstår hva vi sier når vi snakker til den, og den kan styre lys og dører i hjemmene våre. Butikker, sykehus, transport og infrastruktur er avhengig av datamaskiner, og i vinter så vi hvordan en datafeil stoppet all flytrafikk over Norge. Bilene er blitt så avanserte datamaskiner at de snart kan kjøre helt på egen hånd. GPS gjør det enkelt å navigere, men gjør oss også utsatt for overvåking. I det senere har vi også sett hvordan myndigheter kan bruke ansiktsgjenkjenning og avanserte algoritmer til å holde kontroll på innbyggerne sine.

Den digitale utviklingen har helt klart hatt stor påvirkning på samfunnet. Samtidig har regnekraften gjort det mulig for fysikere og ingeniører å gjøre beregninger og utvikle modeller som tidligere ville vært svært krevende, og programmering har blitt en stadig større del av også en fysikers hverdag. I mekanikk er det mulig å gjøre presise beregninger av bevegelse og krefter, der det ikke er nødvendig å se vekk fra for eksempel luftmotstand og friksjon, og en kan simulere store konstruksjoner og materialers holdbarhet. I termofysikk kan en gjøre avanserte beregninger på partiklers bevegelse eller varmeoverføring, mens det i elektromagnetisme er mulig å beregne komplekse felt fra ulike ladningsfordelinger. I kvantemekanikk har det blitt mulig å løse Schrödingerlikningen og beregne energitilstander for komplekse og sammensatte partikkelsystemer. I meteorologi og klimaforskning spiller datamaskinen en avgjørende rolle for å utvikle stadig bedre værvarsler og modeller for globale klimaendringer.

Det er viktig at skolen klarer å følge denne utviklingen på en god måte. I NOU 2015: 8 (2015, s. 36) påpeker Ludvigsenutvalget det store verktøyfokuset den grunnleggende ferdigheten «digitale ferdigheter» har i dagens læreplan. I dag lærer elevene i stor grad å *bruke*

en rekke digitale verktøy, mens det bare er i noen få valgfrie fag, som Teknologi og forskningslære, Informasjonsteknologi og Programmering valgfag, at elevene lærer hvordan programmene fungerer gjennom programmering og algoritmeforståelse. Det kan betraktes som både et demokratisk og et akademisk problem at elevene ikke forstår den fundamentale virkemåten til teknologien vi omgir oss med.

Da England i 2014 bestemte seg for å innføre programmering i skolen, fulgte raskt mer enn 20 EU-land etter. Også i USA er programmering på vei inn i skolen etter at ingeniørvitenskaplige praksiser («engineering practices») fikk en sentral plass i det nasjonale forskningsrådets rammeverk for grunnopplæring i naturfag (NRC, 2012). De to hovedargumentene som blir brukt for innføringen, er at programmering er en viktig kompetanse i seg selv, som etterspørres av arbeidsmarkedet, og at programmering kan bidra til å utvikle andre viktige ferdigheter som problemløsning og analytisk og logisk tenkning. Dette blir gjerne omtalt som «computational thinking» på engelsk, eller «algoritmisk tenkning» på norsk. Slike ferdigheter er dessuten blant dem som kalles «det 21. århundrets ferdigheter» (Balanskat, Engelhardt og Ferrari, 2017; Bocconi, Chioccariello og Earp, 2018).

Høsten 2017 lanserte den norske regjeringen en strategi for digitalisering av norsk skole. Strategien peker på at elevene skal få økt opplæring i teknologi, og herunder programmering og algoritmisk tenkning, for å svare på samfunnets raske teknologiske utvikling (Kunnskapsdepartementet, 2017). I en undersøkelse blant undervisningsmyndigheter i nordiske land ser Balanskat et al. (2017) etter hvilke argumenter som har vært styrende for å inkludere programmering og algoritmisk tenkning i skolen. I Norge finner de at fokuset har vært på å fremme problemløsningsferdigheter, logisk tenkning og andre sentrale kompetanser, i tillegg til å gi forståelse for teknologiens plass i samfunnet. I Sverige og Finland brukes også argumenter om å fremme rekruttering til IKT-sektoren, programmeringsferdigheter, økt motivasjon for matematikk og utvikling av digitale medborgerskap (Balanskat et al., 2017).

I forbindelse med fagfornyelsen av de norske læreplanene, har det blitt bestemt at det ikke skal innføres et eget fag for teknologi, men at opplæringen skal inngå som en del av de allerede eksisterende fagene (Utdanningsdirektoratet, 2017c). Våren 2018 ble kjerneelementene til de nye fagene presentert. Samtidig ble det slått fast at programmering og algoritmisk tenkning blir en del av fagene matematikk, naturfag og kunst og håndverk (Kunnskapsdepartementet, 2018). Arbeidet med fornying av læreplanene for grunnopplæringen og de gjennomgående fagene i videregående skole har pågått parallelt med arbeidet med denne

oppgaven. Revisjon av programfagene for fysikk påbegynnes i det dette arbeidet fullføres, men det forventes at programmering også integreres i fysikk-undervisningen.

1.2 Oppgavens formål og struktur

Det er slått fast at programmering og «algoritmisk» («computational») tenkning skal inn i de nye læreplanene i matematikk og naturfag, og antakelig også i programfagene for fysikk. Dette er i stor grad bestemt av myndighetene, uten at det foreligger et tydelig ønske fra lærere eller evidens i forskningen på at det kan bidra til bedre læring av de øvrige fagområdene. Denne oppgaven forsøker å møte behov som følger av dette, og tar for seg hvordan programmering kan innføres som et gjennomgående verktøy i fysikkundervisningen. Håpet er at oppgaven kan bidra til en god, gjennomtenkt innføring av programmering, basert på forskning, både i utformingen av nye læreplaner og i gjennomføringen i klasserommene. Oppgaven fokuserer på algoritmisk tenkning og beregningsorientert fysikk på videregående nivå, men tar i liten grad for seg styring av roboter, spill- eller sensorteknologi, som også kan være relevante måter å inkludere programmering i skolefagene.

Oppgaven som helhet har elementer av designbasert forskning, som en kombinasjon av forskning og utvikling, ved at den er forankret i en virkelig utdanningskontekst med tett tilknytning til praksis, fokuserer på utforming og testing av nytt innhold i fysikkfaget, benytter flere metoder, og kan inngå som et ledd i en større iterativ prosess (Anderson og Shattuck, 2012). Oppgaven har en tredelt struktur.

Den første delen er en større litteraturstudie (kapittel 2) med gjennomgang av forskningslitteraturen som eksisterer på området og hvilken rolle programmering og algoritmisk tenkning kan ha i fysikkundervisning. I tillegg til å utgjøre en egen studie, vil denne delen også være det teoretiske grunnlaget for oppgavens empiriske deler.

Den andre delen er en intervju-undersøkelse med tre fysikklærere som intervjupersoner (kapittel 3). Denne delen forsøker å belyse noen læreres synspunkter og tanker om hvordan de ser for seg implementering av programmering i deres undervisning. Jeg ser på hvordan de forstår begrepet «algoritmisk tenkning» og hvilke muligheter og utfordringer de ser for seg at programmering kan gi for læring av fysikk. Denne delen er hentet fra masterarbeidets pilotstudie og er tidligere levert som en eksamensbesvarelse i et emne om forskningsmetoder. For arbeidets helhet gjengis den her med en del justeringer og tilpassinger av innholdet.

Den tredje delen er et utviklingsarbeid (kapittel 4) der jeg, basert på litteraturstudien og intervjuet med lærerne, utformer tre undervisningsopplegg tilpasset dagens læreplan i Fysikk 1 og Fysikk 2. Denne delen tar også for seg erfaringer etter utprøving med elever og elevenes svar på en spørreundersøkelse om hvordan de opplevde timens innhold.

I kapittel 5 forsøker jeg å knytte resultatene i litteraturstudien, intervjustudien og utprøvingen av undervisningsoppleggene sammen i en helhetlig diskusjon. Her forsøker jeg å gi noen anbefalinger til lærere, lærerutdannere og forskere, samt forfatterne av de nye læreplanene i fysikk.

1.3 Problemstilling og forskningsspørsmål

Oppgavens overordnede mål er å undersøke problemstillingen om hvordan programmering og algoritmisk tenkning kan og bør integreres i fysikkundervisningen på videregående nivå i skolen på fysikkfagets premisser. Dette gjøres ved å besvare følgende forskningsspørsmål i henholdsvis litteraturstudien, intervjustudien og utviklingsarbeidet:

1. Hva sier forskningslitteraturen om hvilken rolle algoritmisk tenkning og programmering kan spille i fysikkundervisning på videregående nivå?
    a. Hvordan defineres algoritmisk tenkning og programmering i en fysikk-sammenheng?
    b. Kan programmering bidra til læring av fysikk?
    c. Hvordan kan programmering inkluderes i fysikk på fagets premisser?

2. Hvordan forstår fysikklærere begrepet «algoritmisk tenkning», og hvilke utfordringer og muligheter ser lærerne for programmering i fysikkundervisningen?

3. Hvordan kan et undervisningsopplegg med programmering se ut?
    a. Hvor utfordrende opplever elevene opplegget?
    b. Hva opplever elevene som utfordrende?
    c. Ser elevene relevansen til fysikk?

2 LITTERATURSTUDIE: Programmering og algoritmisk tenkning i fysikk

I dette kapitlet gjør jeg en gjennomgang av forskningslitteratur som er skrevet om programmering og algoritmisk tenkning i fysikkopplæringen. Da dette er et nytt felt i skolen og det foreløpig finnes få erfaringer med undervisning av programmering og fysikk, er det et behov for å samle forskning og erfaring som er gjort på området. Dette kapitlet utgjør derfor en egen studie og en større del av masterarbeidet, samtidig som det også er teorigrunnlaget for de kommende kapitlene. For at leseren enklere skal kunne finne tilbake til innholdet som refereres, henviser jeg også til sidetall i originalkilden der det er hensiktsmessig.

Jeg gjør først rede for hvordan litteraturstudien er gjennomført, om søkestrategi, utvalg av artikler og noen etiske betraktninger i metode-seksjonen, kapittel 2.1. Det eksisterer flere definisjoner og ulike forståelser av hva begrepene programmering, algoritme og algoritmisk tenkning innebærer. I kapittel 2.2 definerer jeg derfor begrepene slik jeg vil anvende dem i denne oppgaven. Jeg tar spesielt for meg den mest sentrale litteraturen som er skrevet om algoritmisk tenkning i naturfag og presenterer et teoretisk rammeverk for hva dette innebærer for undervisningen i klasserommet. I kapittel 2.3 stiller jeg spørsmål om hvorvidt, og eventuelt på hvilken måte, programmering kan bidra til læring av fysikk og ser på hvilken forskning som er gjort på dette området. Hvorvidt programmering kan ha nytte i skolens fysikkfag, avhenger i stor grad av hvilket innhold faget har og hva man ønsker at elevene skal lære. I kapittel 2.4 ser jeg på hvordan programmering gir muligheter for større endringer av hvilke fysikkemner som dekkes i læreplanene. Kapittel 2.5 tar for seg empiriske erfaringer som er gjort og praktiske tips som er gitt for hvordan fysikk bør undervises med programmering og algoritmisk tenkning. Til sist gir kapittel 2.6 en kort introduksjon til sentrale metoder for numeriske beregninger som kan være aktuelle å benytte på videregående nivå. Dette kapitlet har et mer matematisk fokus og fokuserer på de faglige sidene ved programmering og numeriske beregninger i en fysikksammenheng.

2.1 Metode

Fokus og forskningsspørsmål

Utgangspunktet for litteraturstudien har vært behovet for kunnskap og for å sammenfatte tidligere forskning om hvordan programmering og algoritmisk tenkning kan integreres i undervisningen. Etter føringer fra utdanningsmyndighetene, skal programmering integreres i skolefagene, og studien fokuserer derfor på at programmeringen må foregå på fagenes premisser og for fagenes skyld. Studien er hovedsakelig rettet mot fysikk på videregående skole, med utgangspunkt i hva som kan være relevant for innholdet i læreplanene som eksisterer i dag, og det er lagt mindre vekt på hvordan programmering kan anvendes i robot-utvikling, sensorteknologi og annen ingeniørvitenskap som hører bedre hjemme i faget Teknologi- og forskningslære.

Forskningsspørsmålene er:

1. Hva sier forskningslitteraturen om hvilken rolle algoritmisk tenkning og programmering kan spille i fysikkundervisning på videregående nivå?
    a. Hvordan defineres algoritmisk tenkning og programmering i en fysikk-sammenheng?
    b. Kan programmering bidra til læring av fysikk?
    c. Hvordan kan programmering inkluderes i fysikk på fagets premisser?

Søkestrategi og utvalg

Litteraturen som presenteres her er funnet gjennom et større arbeid med litteratursøk. Arbeidet ble påbegynt høsten 2018 og har pågått frem til våren 2019. Det er i stor grad benyttet databaser og søkemotorer for forskningsartikler, hovedsakelig Google Scholar, Scopus og Oria. Sentrale søkeord har vært ulike kombinasjoner av computational thinking, algoritmisk tenkning, physics, fysikk, science, naturfag, mathematics, matematikk, upper secondary school, high school, K-12, videregående skole, programming, programmering, numerical physics, numerisk fysikk, computational physics, beregningsorientert fysikk, education, utdanning, learning, læring, undervisning, introduction, beginner, python. Det er hovedsakelig gjort søk på engelsk og norsk, men til dels også på svensk, dansk og tysk.

En utfordring har vært å avgrense søkene for å finne de relevante artiklene. De mest sentrale søkeordene gitt ovenfor, kan gi flere hundre tusen treff, der bare et fåtall omhandler

programmering og algoritmisk tenkning i en relevant undervisningssammenheng. Mange av artiklene omhandler ren informatikk («computer science»), undervisning som innebærer bruk av teknologi, men som ligger utenfor denne oppgavens fokus, eller helt andre «programmer» i betydningen reformer eller prosjekter. Søkene har blitt noe begrenset ved å filtrere artikler publisert etter 2015. Det er også gjort søk filtrert etter 2018 og 2019. Ettersom dette er et relativt nytt forskningsområde, utelukker dette mye av de eldre, irrelevante treffene. I Scopus har det også vært mulig å filtrere ut artiklene som har «utdanning» som nøkkelord.

Det er treffene på noen av de første søkeresultatsidene som er vurdert, basert på hvorvidt de omhandler programmering, algoritmisk tenkning og læring i en realfaglig undervisnings-sammenheng slik det fremkommer i tittel og utdrag gitt i søkemotoren. Hvis dette er tilfelle, har artikkelen blitt åpnet og vurdert ut fra artikkelens sammendrag og konklusjon, og eventuelt lest i sin helhet.

En betydelig andel av kildene er funnet ved å følge litteraturreferanser både bakover og fremover i tid. Førstnevnte er gjort ved å følge en artikkels sitater og referanseliste, mens Google Scholar og Scopus har et verktøy for å finne nye artikler der kilden er sitert. Artiklene til Weintrop et al. (2016) og Wing (2006) har vært viktige utgangspunkt her.

Det kan se ut til at det er gjort begrenset med forskning på området, og effektene av algoritmisk tenkning i undervisningen er usikre (Balanskat et al., 2017). Det har vært en del fokus på å definere hva «algoritmisk tenkning» faktisk er, den historiske utviklingen og anekdotiske bevis på hvordan det kan påvirke utdanningen og læringen, men foreløpig er det få empiriske studier og erfaringer fra klasserommet – særlig fra fysikk på nivå tilsvarende norsk videregående skole. Grover og Pea (2013, s. 42) skriver at det mangler empirisk erfaring generelt om algoritmisk tenkning, og at det trengs mer forskning for å avdekke vanlige utfordringer og misoppfatninger blant skoleelever, og hvordan slike kan unngås. Det er kanskje heller ikke så uventet, gitt at det enda i liten grad er innført i skolene. Det finnes noe erfaring fra universitetsnivå (For eksempel Malthe-Sørenssen, Hjorth-Jensen, Langtangen og Mørken, 2015; Sørby og Angell, 2012). Selv om dette blir litt for avansert for fysikkelever på videregående skole, bør erfaringer gjort fra førsteårsstudenter på universitetet kunne trekkes ned i skolen og tilpasses elevene. Det finnes også noen relevante erfaringer fra programmering på barneskolen (For eksempel diSessa, 2018; Dwyer, Boe, Hill, Franklin og Harlow, 2013), men en del av disse omhandler roboter, lek og problemløsning med programmering som ikke nødvendigvis er relevant for denne oppgaven (For eksempel Wagner, 1998; Zaharija, Mladenović og Boljat, 2013). Det har derfor vært

nødvendig å også inkludere forskning gjort i matematikk og naturfag og realfaglig programmering noe mer generelt.

Tabell 2-1 viser hvordan de 52 artiklene som er tatt med i litteraturstudien fordeler seg på teoretiske og empiriske artikler og hvilket nivå de retter seg mot. Noe av litteraturen inneholder betydelige både teoretiske og empiriske deler, og artiklene kan rette seg mot nivå som passer for flere eller ingen bestemte nivåer i utdanningssystemet. Ettersom skillet mellom de ulike nivåene i andre land også avviker noe fra det norske, er denne oversikten omtrentlig.

OMHANDLER	ANTALL
Totalt antall artikler	*52*
Teoretiske/Perspektiver	33
Empiriske	23
Grunnskolenivå	13
Videregående nivå	19
Universitetsnivå	14

Tabell 2-1 Oversikt over artiklene som er inkludert i litteraturstudien.

Etiske betraktninger

Selv om en gjennomgang av allerede eksisterende forskning ikke har de samme etiske utfordringene ved innsamling av data som andre typer forskningsmetoder, peker Robson og McCartan (2016, ss. 90-92) på områder som har betydning for litteraturstudiens integritet og etikk. Det handler blant annet om innhentingen og utvalget av kilder, om anvendelsen og tolkningen av andre forskeres arbeid, og i tillegg ærlighet og åpenhet i metode og gjennomføring.

Selv om jeg foran har gjort rede for søkestrategier og fokus ved utvalg av kilder, er det en mulighet for at viktig arbeid er oversett eller at annet har fått større oppmerksomhet enn det burde. Det er ført journal under litteraturstudien, men noen av referansene er funnet mer eller mindre tilfeldig, og det er ingen helt klare kriterier for hvorvidt en referanse skulle følges videre eller forkastes. Det gjør at en litteraturstudie er noe vanskelig å ettergå.

Forskeren som gjennomfører litteraturstudien, leser alltid kildene gjennom sine egne «briller», og det er en fare for forventningsskjevhet eller misforståelser av primærkildene. I verste fall vil en uetisk forsker bevisst fabrikkere eller presentere data feilaktig. Litteraturstudien videreformidler et budskap, har mulighet til å påvirke beslutninger og etablere

eller forsterke en antatt «sannhet». Det ligger dermed et viktig ansvar på forskeren for hvilke kilder som inkluderes i studien, men også hvilke kilder som utelates. På grunn av begrensningen i datagrunnlaget, har jeg i denne studien ikke hatt noe uforbeholdent krav om at artiklene må være fagfellevurdert. Kildenes troverdighet er likevel vurdert, blant annet etter forfatterens status i miljøet, hvor artiklene er publisert og hvor ofte de er referert i annen forskning. De fleste publikasjonene som er inkludert i denne litteraturstudien er i stor grad kvalitative, noe som gjør at funnene i enda større grad er åpne for tolkning. Jeg gir her klare kildehenvisninger slik at det skal være enkelt for leseren å finne igjen arbeidene det refereres fra. For å forsterke etterprøvbarheten, er sidetall ikke bare oppgitt ved direkte sitat. Analysen av artiklene og sammenfatningen av resultatene er gjort basert på journalen og notater som er gjort undervegs og med formål om å besvare forskningsspørsmålene. Innsamlingen og analysen er imidlertid for liten og begrenset til å fullt ut oppfylle det Robson og McCartan (2016, ss. 93-94) kaller en «systematisk litteraturgjennomgang».

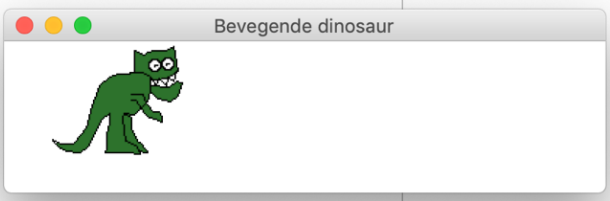
2.2 Programmering og algoritmisk tenkning

Programmering

Vi kan definere *programmering* som å utforme og implementere presise instruksjoner som styrer hvordan en datamaskin skal oppføre seg og reagere på ulike kommandoer. Selve utformingen av dataprogrammet skjer ved at det skrives kode i et *programmeringsspråk*. En datamaskin kan tolke disse kodede instruksjonene og utføre de ønskede handlingene i en bestemt rekkefølge under bestemte betingelser (Rossen, 2017; Sanne et al., 2016, s. 18). For eksempel kan instruksjonen være å skrive ut «De som aldri har gjort en feil, har aldri prøvd noe nytt» til skjermen under betingelsen at brukeren klikker på et bilde av Albert Einstein.

Det finnes ulike typer programmeringsspråk. Språk som for eksempel Scratch og Blockuino er såkalt blokk-programmering. Disse minner litt om puslespill, der barn helt ned i barnehagealder enkelt kan lage morsomme programmer ved å trekke riktige instruksjoner i riktig rekkefølge (se Figur 2-1). Slik får de trening i å strukturere programmene, uten at de selv må skrive avansert kode. Språk som Python og C++ er tekstbasert programmering. Her må koden skrives inn som tekstlinjer, med en streng syntaks. Til gjengjeld er disse bedre egnet for større programmer og kraftigere beregninger.

```python
#Importerer pakker for å animere grafikk
from tkinter import *
import time

#Lager et vindu for animasjonen
vindu = Tk()
vindu.geometry("800x80")
vindu.attributes("-topmost", True)
vindu.title("Bevegende dinosaur")
c = Canvas(vindu, width = 800, height = 80, bg='white')
c.pack()

#Henter bilde av en dinosaur
dinoimg = PhotoImage(file="dinosaur.gif")
dino = Label(image=dinoimg)

#Beregner posisjon numerisk
tidssteg = 0.01
tid = 0
akselerasjon = 1000
fart = 0
xposisjon = 0
yposisjon = 0

dino.place(x=xposisjon,y=yposisjon)
while tid < 4:
    time.sleep(tidssteg)

    tid = tid + tidssteg
    fart = fart + akselerasjon*tidssteg
    xposisjon = xposisjon + fart*tidssteg

    dino.place(x=xposisjon,y=yposisjon)

    vindu.update()


vindu.mainloop()
```



Figur 2-1 Kode for å beregne og animere bevegelsen til en akselererende dinosaur numerisk. Øverst i tekst-programmeringsspråket Python og nederst i blokk-programmeringsspråket Scratch.

Algoritmer

En serie med fullstendige og presise instruksjoner kalles gjerne for en *algoritme*. En algoritme kan være skrevet i kode og utføres av en datamaskin, men kan også være en fremgangsmåte mennesker kan bruke for å løse matematiske oppgaver, for eksempel divisjonsalgoritmen, eller andre typer problemer, som en matoppskrift, en strikkeoppskrift eller en veibeskrivelse. Datamaskinens store styrke er at den kan utføre algoritmer mye raskere og mer presist enn det et menneske klarer (og har lyst til). Det kreves liten eller ingen grad av vurdering og skjønn for å utføre en algoritme. Derimot kan det være en krevende tankeprosess å utvikle en algoritme (Sanne et al., 2016, s. 18; Store norske leksikon, 2018).

Definisjon av algoritmisk tenkning

Kjerneelementgruppen for matematikk mener det er viktig at elevene behersker å utføre algoritmer, men mener det er enda viktigere at elevene forstår hvorfor algoritmene fungerer og hvordan de kan utlede og utforme egne algoritmer (Utdanningsdirektoratet, 2017c). Det å bryte et problem ned til enkle instruksjoner og sette disse sammen til en algoritme, blir på norsk gjerne omtalt som *algoritmisk tenkning*. Det er også dette begrepet som brukes i de første utkastene til den nye læreplanen i matematikk som ble publisert oktober 2018. I beskrivelsen av matematikkfaget defineres algoritmisk tenkning i forbindelse med utforsking og problemløsning: «Algoritmisk tenking er viktig i prosessen med å utvikle strategiar og framgangsmåtar og inneber å kunne bryte ned eit problem i delproblem som kan løysast systematisk» (Utdanningsdirektoratet, 2018b). Det er også omtalt i forbindelse med kritisk tenkning og livsmestring at algoritmer har stor påvirkning på oss i hverdagen: «(…) det krev forståing for algoritmisk tenking og den påverknaden vi blir utsette for i dagleglivet» (Utdanningsdirektoratet, 2018b). I læreplanskissene for naturfag, nevnes ikke begrepet algoritmisk tenkning eksplisitt, men programmering er omtalt som en digital ferdighet i faget, og flere egenskaper ved algoritmisk tenkning er omtalt i problemløsning med teknologi: «Gjennom praktisk problemløsning skal elevene modellere, konstruere, forstå, bruke og utforske teknologi knyttet til hverdagssituasjoner og innovasjon» (Utdanningsdirektoratet, 2018a).

Begrepet «algoritmisk tenkning» stammer fra det engelske «*computational thinking*». I mangel av en bedre norsk oversettelse, brukes «algoritmisk tenkning» også som en direkte oversettelse av «computational thinking» på norsk. En svakhet ved denne oversettelsen, som også brukes i

tilsvarende form på finsk (Balanskat et al., 2017), er at det kan forveksles med det engelske begrepet «algorithmic thinking», som på engelsk har en mer snever definisjon enn «computational thinking». «Algorithmic thinking» er nærmere knyttet til koding og algoritmer alene, slik også Utdanningsdirektoratet bruker det i læreplanskissen for matematikk, mens «computational thinking» også innebærer en bredere forståelse av databehandling, modellering og datamaskiner generelt. Futschek (2006, s. 160) definerer «algorithmic thinking» ved seks ferdigheter knyttet til å lage og forstå algoritmer:

- evnen til å analysere gitte problemer
- evnen til å spesifisere et problem presist
- evnen til å finne de grunnleggende prosessene som er adekvate for det gitte problemet
- evnen til å konstruere en riktig algoritme for et gitt problem ved å bruke de grunnleggende prosessene
- evnen til å tenke på alle mulige spesielle og vanlige tilfeller av et problem
- evnen til å forbedre effektiviteten til en algoritme
  (Futschek, 2006, s. 160, min oversettelse)

Dette er, som vi skal se, noe snevrere enn «computational thinking». Vi kan si at «algorithmic thinking» bare utgjør en del av «computational thinking».

Andre norske begrep som forekommer i litteraturen, er «beregninger», «beregningsorientering» og «beregningsorientert fysikk» (Malthe-Sørenssen et al., 2015; Sørby, 2010), som vi kunne ha forlenget til «beregningsorientert tenkning». Å oversette «computational» med «beregningorientert» (eller «numerisk») er heller ikke spesielt heldig i denne sammenhengen, da dette gjør begrepet snevrere mot numeriske løsningsstrategier.

«Computational thinking» kunne muligens oversettes med «digital tenkning», selv om dette kanskje kan forveksles med kunstig intelligens eller nettvett og digital dømmekraft. På svensk er det vanlig å anvende begrepet «datalogiskt tänkande» (Kjällander, Åkerfeldt og Petersen, 2016; Stolpe, 2018). Begrepet «datalogi» står også oppført i norske ordbøker som et sjeldent brukt synonym til «informatikk» og «IKT». Likevel kan det vurderes om ikke «computational thinking» heller bør oversettes til «datalogisk tenkning» på norsk, der en sammentrekning av «data» og «logikk» kanskje bedre beskriver begrepets innhold. Ettersom

dette begrepet foreløpig ikke anvendes i Norge, vil jeg i fortsettelsen benytte «algoritmisk tenkning» som en norsk oversettelse av «computational thinking».

Begrepet «*computational thinking*» ble første gang brukt av Seymour Papert (1980). Papert pekte på at når barn bruker datamaskinen i undervisningssammenheng, er det gjerne maskinen som styrer interaksjonen ved å gi oppgaver av passe vanskelighetsgrad, tilbakemeldinger og informasjon. «Datamaskinen programmerer barnet» (Papert, 1980, s. 19, min oversettelse). Han så for seg en undervisning der det i stedet er barnet som skal bestemme hvordan datamaskinen skal oppføre seg, og at dette igjen vil påvirke hvordan barnet selv tenker – også når datamaskinen ikke er i nærheten (Papert, 1980, ss. 4, 19). Det er slik algoritmisk tenkning skiller seg fra det som i dagens læreplan heter «digitale ferdigheter», hvor fokuset i større grad ligger på å *bruke* teknologi, enn på å utforme og forstå virkemåten til digitale verktøy (Sanne et al., 2016, ss. 43-44). Wing (2006) forsøkte å oppsummere hva «computational thinking» er, og hva det ikke er. Hun ser på det som en grunnleggende ferdighet som må sidestilles med lesing, skriving og regning, og en slik tenkemåte innebærer «å løse problemer, utforme systemer og forstå menneskelig atferd basert på fundamentale prinsipper i datavitenskap» (Wing, 2006, s. 33). Hun påpeker at det er en måte *mennesker* tenker på, ikke datamaskiner, og at det inkluderer mer enn bare programmering. Vi kan følge Grover og Pea (2013, s. 39) og si at hovedessensen i «computational thinking» er å løse et problem ved å tenke som en informatiker.

Wings definisjon av «computational thinking» var imidlertid ikke helt entydig, og det har i ettertid blitt skrevet flere artikler som forsøker å gi en mer presis definisjon av hva «computational thinking» faktisk er, både innholdsmessig og pedagogisk. Barr og Stephenson (2011, s. 52) har laget en oversikt over sentrale sider ved «computational thinking». Disse innebærer: datasamling, datanalyse, datarepresentasjon, dekomposisjon av problemer, abstraksjon, algoritmer og prosedyrer, automatisering, parallellisering og simulering. De gir også eksempler på hva hvert av disse elementene kan innebære i ulike fagdisipliner. Grover og Pea (2013, ss. 39-40) oppsummerer de elementene de nå mener har blitt akseptert å utgjøre «computational thinking» slik: Abstraksjoner og mønstergeneralisering, systematisk informasjonsbehandling, symbolsystemer og representasjoner, algoritmisk forståelse av kontrollflyt, strukturert dekomposisjon av et problem, å tenke iterativt, rekursivt og parallelt, betinget logikk, effektivitets- og ytelsesbegrensninger, systematisk feilsøking og feilretting. Grover og Pea (2013, s. 40) fremhever at programmering inngår i «computational thinking» som både en ferdighet og et verktøy.

Wing har senere forsøkt å gi en tydeligere definisjon av hva «computational thinking» er: «Computational thinking er tankeprosessen som inngår i å formulere et problem og uttrykke dets løsning(er) på en slik måte at et menneske eller en datamaskin kan utføre beregningen effektivt» (Wing, 2014, min oversettelse). diSessa (2018, s. 19) kritiserer imidlertid Wing for ikke å løfte algoritmisk tenkning fra ren datavitenskap, slik at det kan bygges inn i andre fagdisipliner, som fysikk. Selv bruker han begrepet «computational literacy». Han presiserer imidlertid at «computational literacy», til tross for likheter til «computational thinking», ikke er synonyme begreper (diSessa, 2018, s. 17).

Når dette masterarbeidet nå nærmer seg slutten, har Utdanningsdirektoratet (2019a) presisert at de forstår begrepet «algoritmisk tenkning» som en norsk oversettelse av «computational thinking». Utdanningsdirektoratets beskrivelse av «algoritmisk tenkning» samsvarer dermed godt med forståelsen til Wing (2014):

> Å tenke algoritmisk er å vurdere hvilke steg som skal til for å løse et problem, og å kunne bruke sin teknologiske kompetanse for å få en datamaskin til å løse (deler av) problemet. I dette ligger også en forståelse av hva slags problemer/oppgaver som kan løses med teknologi og hva som bør overlates til mennesker. (Utdanningsdirektoratet, 2019a)

Algoritmisk tenkning beskrives altså ikke bare som en problemløsningsstrategi, men også som en kompetanse i bruk av digitale verktøy. Å bryte et problem ned i mindre, enklere løsbare problemer, håndtere informasjon, logisk analysere, abstrahere, generalisere, modellere, forutse og vurdere, trekkes frem som sentrale nøkkelbegrep ved algoritmisk tenkning. Figur 2-2 gjengir hvordan direktoratet oppsummerer nøkkelbegrepene og arbeidsmetodene som kjennetegner «den algoritmiske tenkeren».

Figur 2-2 Nøkkelbegreper og arbeidsmåter som kjennetegner «den algoritmiske tenkeren».
KILDE: Utdanningsdirektoratet (2019a)

Algoritmisk tenkning i matematikk og naturfag

Weintrop et al. (2016) har utviklet en taksonomi for algoritmisk tenkning. Den samsvarer i stor grad med de tidligere definisjonene av algoritmisk tenkning, men er spesielt utformet for matematikk og naturfag. Taksonomien er i basert på et rammeverk for amerikansk grunnopplæring i naturfag fra det nasjonale forskningsrådet i USA (NRC, 2012). Dette vektlegger natur- og ingeniørvitenskapelige «praksiser» i opplæringen (Bybee, 2011), og Weintrop og kollegenes taksonomi for algoritmisk tenkning har fokus på anvendelse i klasserommet. Denne er gjengitt i en norsk oversettelse i Figur 2-3.

Den første kategorien, «datahåndtering», handler om å samle, generere, behandle, analysere og visualisere data. Elevene *samler* data når de gjør målinger og observasjoner og lagrer resultatene, gjerne ved hjelp av et digitalt verktøy. De kan for eksempel bruke en datalogger til å måle posisjonen til en lekebil eller en kraftmåler til å finne kraften i en fjær som strekkes.

Datahåndtering	Modellering og simulering	Digital problemløsning	Systemtenkning
Samle data	Bruke digitale modeller for å forstå et begrep	Forberede problemer til å løse dem digitalt	Undersøke et komplekst system som en helhet
Generere data	Bruke digitale modeller til å finne og teste løsninger	Programmere	Forstå sammenhenger innad i et system
Behandle data	Vurdere digitale modeller	Velge effektive beregningsverktøy	Tenke i nivåer
Analysere data	Utforme digitale modeller	Vurdere ulike tilnærminger/løsninger til et problem	Kommunisere informasjon om et system
Visualisere data	Implementere digitale modeller	Utvikle modulære digitale løsninger	Definere systemer og håndtere kompleksitet
		Lage digitale abstraksjoner	
		Feilsøking og feilretting	

Figur 2-3 Weintrop et al. (2016, s. 135) sin taksonomi for algoritmisk tenkning i matematikk og naturfag. Min oversettelse til norsk.

Noen ganger arbeider man imidlertid med fenomener som er vanskelige å observere eller måle. Eksempler kan være store avstander i verdensrommet, partiklers virrevandring i en gass eller magnetisk felt rundt en stavmagnet. Da kan det være nødvendig at elevene *genererer* dataene for å undersøke fenomenene. I mange tilfeller vil det være nødvendig at elevene *behandler* de innsamlede dataene før de kan analyseres eller presenteres. Det kan innebære å sortere eller filtrere dataene, men det kan også være å formatere dataene slik de for eksempel kommer fra dataloggeren, til et format som kan håndteres i programmeringsspråket. Digitale verktøy kan være til god hjelp når en skal *analysere* dataene. Elevene kan for eksempel lage et program som kategoriserer tusenvis av datapunkter, eller de kan finne mønstre og trender med regresjon i GeoGebra. Til slutt er digitale verktøy svært godt egnet til å *visualisere* og presentere dataene. Det kan være å lage en graf eller et diagram over dataene, men det kan også være å lage en dynamisk animasjon. Denne kan også lages interaktiv, slik at brukeren selv kan samhandle med

dataene. Slik kan eleven arbeide med større og mer virkelighetsnære datamateriale, og de kan skaffe seg en bedre oversikt over informasjonen som ligger i dataene.

Praksisene som omhandler «modellering og simulering», innebærer å bruke, vurdere, utforme og konstruere digitale modeller. Det vektlegges at elevene ikke bare skal anvende modeller, men også kunne vurdere en modell, hvilke forenkleringer som er gjort og dens gyldighetsområde, og elevene skal selv kunne utforme digitale modeller som senere implementeres, for eksempel i et programmeringsspråk.

«Digital problemløsning» handler om problemløsningsstrategier med digitale verktøy og utgjør stort sett det vi kan kalle den «snevrere» forståelsen for algoritmisk tenkning, i samsvar med definisjonen til Futschek (2006). Her inngår praksiser som å kunne bryte ned problemer i mindre deler og omforme dem slik at de kan løses digitalt, å kunne utarbeide algoritmer, programmere egne programmer eller tilpasse og forstå andres programmer og algoritmer. Det handler dessuten om å vurdere ulike løsningsstrategier og å velge de riktige verktøyene. Elevene lærer å utvikle moduler, det vil si mindre komponenter eller funksjoner, som kan kjøres alene eller gjenbrukes i det samme eller fremtidige problemer, og å abstrahere og generalisere problemene. Innen denne kategorien inngår også feilsøking og feilretting, som vil si å finne ut hva som gikk eller eventuelt kan gå feil, og å rette det opp.

Til sist i taksonomien kommer «systemtenkning». Det arbeides med systemer i flere deler av fysikken, for eksempel åpne eller isolerte systemer i termofysikk, sammensatte systemer i mekanikk, eller referansesystemer i relativitetsteori. Et system kan betraktes på mikronivå, som for eksempel enkeltpartiklers bevegelse i en gass, eller makronivå, der en betrakter størrelser som gassens temperatur eller varmeoverføringer. Systemtenkning brukes også i andre fagdisipliner, og med den digitale utviklingen har systemene fått en enda mer sentral rolle, blant annet i oppbyggingen av større dataprogrammer og teknologiske systemer. Weintrop et al. (2016) ser på systemtenkning som en problemløsningsstrategi, men som skiller seg fra den tradisjonelle problemløsningen. I stedet for å bryte et problem ned i mindre, løsbare problemer, handler systemtenkningen om å se sammenhenger mellom mindre bestanddeler og å forstå et komplekst system som helheten det utgjør. Det handler om å forstå og kommunisere hvordan de enkelte komponentene samvirker, tenke i nivåer og å definere og håndtere kompleksitet.

2.3 Programmering og læring av fysikk

At programmering skal inkluderes i de øvrige fagene i skolen, er bestemt av myndighetene, og vi har sett at denne modellen innføres i flere land. Spørsmålet er da om det finnes evidens i forskningen om at programmering kan bidra til læring av det som tradisjonelt har vært innholdet i fysikk, eller om programmering hovedsakelig inkluderes som en metode for å lære andre aspekter ved faget. Jeg har funnet begrenset med forskning som gir noen tydelig sammenheng mellom programmering og læring av fysikk, hvilket samsvarer med funnene Barcelos, Munoz, Villarroel, Merino og Silveira (2018) og Hsu, Chang og Hung (2018) gjør i to systematiske litteraturgjennomganger om læring gjennom algoritmisk tenkning. Vi har sett at det finnes noen visjonære tanker om hvordan programmering og algoritmisk tenkning bør integreres som en grunnleggende ferdighet, i likhet med lesing, skriving og regning (diSessa, 2018; Wing, 2006), og det er gjort enkelte empiriske undersøkelser med programmering i sammenheng med læring av fysikk, men det er få studier som kan konkludere om læringseffektene av programmering.

For å kunne si noe om programmering kan gi økt læring i fysikk, må vi først avklare hva fysikk faktisk er, og hva elevene skal lære. Det er helt klart at programmering og numeriske beregninger benyttes av fysikere i både forskning og industri, og det er dermed mulig å argumentere for at programmering i seg selv er et så viktig verktøy i fysikk, at elevene bør lære det med en gang. Det er derimot ikke gitt at elevene i skolen bør introduseres til numerisk fysikk før de har arbeidet mer kvalitativt med begreper og fenomener. Jeg vil komme tilbake til dette spørsmålet i neste delkapittel, der jeg ser på hva forskning sier om hvordan programmering kan påvirke innholdet i fysikkfaget. Her forholder jeg meg til hvordan programmering kan bidra til økt forståelse og læring av begreper og fenomener i en mer tradisjonell forstand, og jeg forsøker å knytte programmering til sentrale områder i fysikkdidaktikk.

Programmering som fremmer fysikklæring

Taub, Armoni, Bagno og Ben-Ari (2015) har i en kvalitativ og kvantitativ undersøkelse sett etter hvilke *kunnskapsintegrerende prosesser* som kommer til syne når elever på videregående nivå arbeider med programmering i fysikk. De kunnskapsintegrerende prosessene de så etter, var å *fremkalle* ideer, *legge til* nye ideer, *utvikle kriterier* for å vurdere ideer, og *å finne ut og reflektere*. De finner fire områder ved programmering som de mener fremmer kunnskapsintegrerende prosesser, og dermed bidrar til konseptuell fysikklæring (Taub et al., 2015, ss. 20-21):

1) *Strukturkunnskap* omhandler programmets struktur og definisjon av variable og objekter som beskriver fysiske objekter og størrelser. Dette krever at elevene fremkaller fysikkunnskapene sine, lager kriterier for å velge nyttige navn og riktige egenskaper ved de fysiske objektene, og finner ut objektenes struktur.
2) *Prosedyrekunnskap* omfatter de underliggende prosedyrene som utfører simuleringen og setter objektene til liv. Det kan være å håndtere inn- og utdata, kontrollstrukturer, løkker og funksjoner. Elevene må fremkalle tidligere kunnskap om fenomenet de skal simulere, de må vurdere om lovene og formlene de finner er relevante for simuleringen, de oppdager kanskje at de må innhente ny kunnskap for å kunne løse problemet, og de forsøker å finne ut hvordan formlene kan integreres i prosedyren og oversettes til programkode.
3) *Systemkunnskap* handler om det systemet som simuleres og hvordan enkelte objekter påvirker hverandre og systemet som helhet. Også her observerer de at elevene fremkaller kunnskap de allerede har, legger til nye ideer, utvikler kriterier for å vurdere hvilke egenskaper som hører til hvilke objekter, og finner ut av hvordan de ulike objektene forholder seg til hverandre.
4) *Iverksettelsen* er det siste området ved programmering, og det starter når elevene kjører programmet sitt. Her reflekterer de over den fysikkunnskapen som ligger i simuleringen, og alle de kunnskapsintegrerende prosessene kan igjen komme til syne. Den ferdige simuleringen kan bekrefte elevenes allerede kjente kunnskap, eller den kan gi ny forståelse av fenomenet. Dessuten må elevene vurdere og reflektere over simuleringen, og muligens finne og rette feil.

Taub et al. (2015) finner altså områder ved programmering der kunnskapsintegrerende prosesser kommer til syne, som igjen kan tyde på at læring forekommer. Programmering utgjør da mer enn bare et digitalt verktøy. De sammenlikner derimot ikke effektene av denne læringen med andre former for (tradisjonell) fysikkundervisning.

Vi har tidligere sett hvordan algoritmisk tenkning kan anses som en problemløsningsstrategi, som handler om å bryte ned problemene i enkle deler, for så å sette dem sammen igjen som et helhetlig system. Hvis elevene med programmering trenes i en slik tenkemåte, kan det være at elevene blir flinkere til å strukturere ideer og tankeprosesser, stille mer relevante, gjennomtrengende spørsmål, avgjøre hvilke egenskaper som er relevante for problemet og til å se sammenhenger mellom ulike representasjonsformer (Sadiku, Shadare og Musa, 2017, s. 427; Taub et al., 2015, s. 11).

Programmering eller algebra som fysikkens språk

Sherin (2001) har studert hvordan programmering kan brukes i stedet for algebra som «fysikkens språk». Han har en formodning om at programmeringsspråket kan være lettere å forstå enn tradisjonell algebra, ved at en student kan gå gjennom koden, linje for linje, og gjøre nøyaktig de same operasjonene som datamaskinen vil gjøre. Han peker på at hvis dette er tilfelle, vil tid elevene i dag bruker på å sette tall inn i formler uten å forstå fysikken bak, kunne byttes ut med aktiviteter som faktisk fremmer forståelse og læring. Den andre formodningen han fremmer, er at programmeringsspråket kanskje kan egne seg til å representere andre typer fenomener enn det algebra kan. Hvis dette er tilfelle, kan en da ikke snakke om hvorvidt programmering kan gjøre jobben *bedre* enn algebra, men snarere hvordan de gjør jobben forskjellig, og om programmeringsbasert fysikk også er en akseptabel form for fysikk (Sherin, 2001, s. 4).

Studien ble gjennomført blant universitetsstudenter på tredje semester. En gruppe ble bedt om å løse tradisjonelle algebraiske fysikkoppgaver, mens en annen gruppe arbeidet med programmering. Studentene hadde allerede studert fysikk en tid, og de hadde dermed fått opplæring i algebraisk fysikk fra før. Det gjør at sammenlikningen mellom en algebraisk tilnærming og programmeringstilnærming kan være noe forskjellig fra det en vil finne dersom elevene lærer fysikk gjennom programmering for første gang. Likevel bør vi kunne anta at algebra også vil få en sentral rolle i et fysikklasserom som anvender programmering, og studien kan slik gi informasjon om hvordan programmering kan bidra til læring av fysikk. Først observerer Sherin (2001, ss. 30-31, 43) at måten studentene konstruerer og tolker en programmeringslinje i stor grad likner på måten de konstruerer og tolker et algebrauttrykk. Han finner også argumenter både for og mot at programmeringskode faktisk er enklere å tolke enn algebraiske uttrykk. Når studentene løser et problem algebraisk, kan de beholde bokstaver for konstanter og verdier, mens i programmering må elevene velge numeriske verdier. Det gjør at programmering kanskje er bedre egnet til å beskrive konkrete eksempler, mens algebra er bedre egnet for generalisering (Sherin, 2001, s. 42). Måten programmeringsspråk kun utfører én linje av gangen, gjør videre at studentene tvinges til å plassere operasjonene i en rekkefølge. Slik tror Sherin (2001, ss. 50, 54) at programmering kan bidra til en økt forståelse av prosess og kausalitet – årsak og virkning, mens algebra er bedre egnet til å regne på tilfeller i balanse og likevekt. Sherin (2001, s. 55) tar ikke stilling til om programmering vil bidra til økt læring eller hjelpe til med å forstå algebraen, men konkluderer med at innføring av programmering kan påvirke *hva* som læres, hvilket jeg kommer tilbake til i kapittel 2.4.

Programmering til å utvikle modeller

Modeller har vært et sentralt emne innenfor fysikkdidaktisk forskning i en årrekke, og som vi ser av blant annet taksonomien til Weintrop et al. (2016), utgjør modeller også en viktig del av algoritmisk tenkning. I prosjektet FYS21 implementerte Angell, Kind og Henriksen (2008) empirisk-matematisk modellering i fysikk på videregående skole. De taler for å bruke modeller i fysikk-undervisningen med et *faglig* og et *pedagogisk* argument: Fysikkens egenart innebærer å modellere virkeligheten, utvikle og anvende modeller. Dessuten kan modeller være nyttige for å undervise, lære og forstå fysiske begreper, ulike representasjonsformer og sammenhengene med eksperimenter (Angell, Kind og Henriksen, 2008, s. 114). I intervju med lærere kom det frem at modellering og praktisk arbeid i fysikk kan bidra til elevers motivasjon, forståelse av den historiske utviklingen av fysikken og forståelse av modellering som vitenskapelig metode. Dessuten er modeller godt egnet i undersøkende læring. Noen av oppgavene i prosjektet var å modellere sammenhenger mellom kraft og forlengelse ved strekking av seigmenn og mellom masse og luftmotstand ved fallende muffinsformer (Angell, Kind, Henriksen og Guttersrud, 2008). De finner at elevenes forståelse av naturvitenskapens egenart, deres læringsstrategier og deres evne til å håndtere ulike representasjonsformer styrker hverandre, og de tror at elever blir i bedre stand til å forstå fysikkinnhold hvis de har gode læringsstrategier (Angell, Kind, Henriksen, et al., 2008, s. 263). Med programmering som verktøy, åpner man for å gi modellering og simulering i fysikkundervisningen en ny dimensjon.

Wilensky, Brady og Horn (2014) bruker såkalt «agentbasert modellering», som betyr at en enkelt «agent» i et større system utstyres med visse egenskaper. Så settes disse agentene sammen slik at de beskriver et større system. På den måten kan man ved å modellere en agents egenskaper på mikronivå, studere resultatene på makronivå. I deres eksempel, fra naturfag, modellerer de et økosystem bestående av gress, sau og ulv. Hver av agentene, f.eks. en sau, får visse egenskaper, som å spise gress, å sulte i hjel eller å formere seg. Når mange sauer settes sammen med gress og ulver, kan populasjonsutviklingen i økosystemet beskrives på makronivå. Et annet eksempel de trekker frem, er hvordan atomer i en gass på mikronivå kan utstyres med visse egenskaper, som fart og kollisjon, mens man på makronivå kan studere trykk og temperatur. Slik kan det tenkes at agentbasert programmering kan bidra til å skape en større forståelse for sammenhenger mellom egenskaper på mikronivå og makronivå (Wilensky et al., 2014, s. 26).

Caballero et al. (2013, s. 38) hevder numerisk modellering i fysikk har flere pedagogiske fordeler. For eksempel kan numeriske beregninger bidra til å forsterke sammenhengen mellom krefter og bevegelse. Ved hvert tidspunkt beregner elevene kraftsummen på et legeme, og med

Newtons 2. lov kan de da beregne akselerasjonen, farten og posisjonen et lite tidssteg senere. Deretter gjentas beregningen av ny kraftsum og ny posisjon helt til hele bevegelsen er beregnet (dette er i praksis Eulers metode, som beskrevet i kapittel 2.6). Matematikken er relativt enkel og kan brukes til å beskrive bevegelser langt mer generelt enn det en analytisk tilnærming vil kunne gjøre, og dermed kan elevene modellere mer virkelighetsnære problemer. Caballero et al. (2013, s. 39) mener dette i større grad knytter sammen kraft og bevegelse og øker forståelsen for hvordan krefter påvirker bevegelsen. Dette samsvarer med det Sherin (2001) finner om at programmering egner seg til å beskrive kausalitet. Dessuten er veien kort til å gjøre små justeringer på krefter og startbetingelser for å undersøke hvordan det påvirker bevegelsen. Endimensjonale bevegelser kan utvides til flere dimensjoner med de samme prinsippene. Dessuten er det relativt enkelt å visualisere beregningene med grafer og animasjoner. Tallene kommer til liv, og beregningene visualiseres umiddelbart. Caballero et al. (2013, s. 39) mener dette kobler ulike representasjonsformer og modeller tettere sammen.

Hutchins et al. (2018) gjennomførte et undervisningsopplegg om krefter og friksjon med elever på videregående nivå. Elevene hadde brukt Newtons første lov til å beregne at den statiske friksjonen måtte senkes til «500» (*sic*) for at en pakke skulle bevege seg. Når de la inn dette i koden, oppdaget de umiddelbart at noe var galt, siden pakken fortsatt ble liggende i ro. Elevene startet da å diskutere, og kom frem til at friksjonen måtte senkes til «499,9» for akkurat å sette i gang akselerasjonen. Hutchins et al. (2018, s. 161) påpeker at hvis elevene hadde gjort den samme oppgaven på papir, ville denne diskusjonen sannsynligvis ikke oppstått, fordi de ikke hadde fått den umiddelbare responsen som de får ved å kjøre programmet. Det viser at programmering kan bidra til å avdekke misoppfatninger. Eksempelet viser imidlertid også at elevene ikke regner med enheter, som kanskje er en naturlig konsekvens av at programmeringsspråket kun regner med tallverdiene.

Kognitiv overbelastning

Taub et al. (2015, ss. 11, 22) peker på noen uheldige sider ved å benytte programmering i fysikkundervisningen. Blant annet kan det være tidkrevende, spesielt hvis elevene ikke kan programmering fra før, og det kan bli en kognitiv overbelastning når elevene må kombinere flere fagområder som oppleves vanskelige for seg. Det finnes en rekke studier om at å lære seg programmering ofte oppleves vanskelig for nybegynnere (Se for eksempel Jenkins, 2002; Kelleher og Pausch, 2005; Lahtinen, Ala-Mutka og Järvinen, 2005; Piteira og Costa, 2013). Caballero et al. (2013, s. 41) opplevde i et fysikk-kurs på videregående nivå at elevene hadde

problemer med å finne feil og avgjøre hvorvidt de hadde gjort en kodefeil eller om det var fysikk-tankegangen som var feil.

Det kan understøttes av funnene Sørby og Angell (2012, ss. 288-290) gjør av at fysikkstudentene arbeider i atskilte moduser – fysikkmodus, matematikkmodus og programmeringsmodus – avhengig av oppgaven de arbeider med. Det er utfordrende for studentene å kombinere kunnskaper de har lært tidligere i adskilte kontekster i en fysikkfaglig sammenheng. Når programmering inkluderes i tillegg til matematikk, blir det enda mer som må kombineres på samme tid. Programmeringsmodusen kjennetegnes ved at studentene i større grad benytter en prøve- og feile-strategi. I stedet for å sjekke om den bakenforliggende fysikken er riktig, leter de etter feil i koden og de fokuserer på hvordan kode har blitt skrevet i liknende problemer tidligere. De glemmer det aktuelle problemet de jobber med nå. Studentene går også i gang med å programmere uten noen plan, og de lar resultatet av koden bekrefte om det de tenker er riktig eller ikke. Programmeringen ser altså ut til å gjøre det mindre skummelt å gjøre feil, og senker terskelen for å komme i gang. Det kan imidlertid diskuteres om det kanskje er så enkelt at studentene ikke engang tenker selv før de prøver, og at de ender opp med rotete kode og mindre forståelse for hva de faktisk forsøker å beregne.

Sadiku et al. (2017) mener at de tette båndene mellom fysikk, matematikk og informatikk kan gjøre det utfordrende å velge ut emner som skal dekkes i et fysikk-kurs med programmering på universitetsnivå. Det kan være nødvendig å bruke tid på emner som egentlig hører hjemme i matematikk eller et eget programmeringsfag, ettersom elevene kanskje ikke kan nok til å bruke det i en fysikk-sammenheng. Det tilsvarende vil sannsynligvis også gjelde på lavere nivå. Caballero et al. (2013, s. 41) erfarte at deres niendeklassinger måtte bruke mye tid på å lære programmeringsspråket på nytt, siden det ofte gikk for lang tid mellom hver gang elevene arbeidet med det. Selv om det er viktig å bruke det ofte nok til at elevene blir trygge på verktøyet, er de også tydelige på at fysikk-kurset ikke må inneholde mer programmering enn nødvendig. Det er fysikk-timer, og ikke informatikk (Caballero et al., 2013, s. 39). Også Yaşar (2013, ss. 16-17) peker på at begrenset med tid og tilgang på teknologisk utstyr, en allerede tett læreplan i den amerikanske videregående utdanningen og bratte læringskurver for både lærere og elever, er utfordringer ved innføring av programmering og modellering i skolen.

2.4 Beregninger åpner for nye måter å bygge opp fagene

Når diSessa (2018, s. 5) omtaler innføring av programmering og digital kompetanse i skolen, drar han historiske paralleller. På samme måte som det arabiske tallsystemet forenklet regneoperasjoner betydelig sammenliknet med romertallsystemet og gjorde regning allment tilgjengelig, mener han at programmering fundamentalt kan endre læring i matematikk og realfag. Han viser hvordan Galileo Galilei brukte linje på linje med tekst for å bevise at forholdet mellom distanser og forholdet mellom tidene brukt på disse distansene, er like for en partikkel med konstant fart. Dette var før algebraens tid. Det samme beviset kunne vi i dag gjort med bare et par linjer. På samme måte som algebra i noen tilfeller kan utkonkurrere ord, mener han programmering og digitale beregninger vil bli viktigere enn algebra, fordi det er mer generelt nyttig og prinsipielt enklere å lære seg.

diSessa (2018, s. 12) gjennomførte et opplegg om bevegelse i matematikk med sjetteklasseelever. Da han søkte om støtte til å undervise vektorregning, fikk han avslag med begrunnelse om at det er «absurd ambisiøst» og «uegnet» å undervise dette på 6. trinn. Da han likevel gjennomførte opplegget, med en tilnærming gjennom visualisert programmering, viste det seg å være en «braksuksess». Sjetteklassingene kunne fint arbeide med vektorregning. Det viser at man med nye verktøy og ny digital kompetanse, må tørre å tenke nytt om hvilket innhold som undervises. Der man tidligere har startet med forenklede spesialtilfeller for å kunne løse likninger analytisk, for eksempel å se vekk fra luftmotstand, anta konstant akselerasjon eller sirkulære planetbaner, kan man med en numerisk tilnærming starte med de generelle tilfellene. Spesialtilfellene og de analytiske løsningene kan studeres nærmere senere. En diskret tilnærming til for eksempel derivasjon og integrasjon, er ifølge diSessa langt mer generaliserbar enn en algebraisk og analytisk tilnærming, der for eksempel «stigningstallet til en tangent», eller «arealet under en kurve», heller bidrar til å skjule den fundamentale forståelsen (diSessa, 2018, s. 12).

Universitetet i Oslo utmerker seg med sin oppbygning av fysikkstudiet og tette integrasjon til numeriske beregninger (Malthe-Sørenssen et al., 2015). I første semester tar fysikkstudentene emner i grunnleggende analyse, modellering, beregninger og programmering, som følger hverandre tett. Malthe-Sørenssen et al. (2015, s. 306) eksemplifiserer med at elevene den første uken lærer den matematiske definisjonen av derivasjon, før de neste uke lærer den numeriske definisjonen, og til slutt implementerer derivasjonsalgoritmen i et programmeringsspråk den tredje uka. Når studentene har sitt første fysikkemne, mekanikk i andre semester, er dette bygget opp med utgangspunkt i de sentrale lovene i fysikk og de

generelle numeriske teknikkene. Slik trenger en ikke lenger ta hensyn til at de utvalgte problemene må resultere i analytisk løsbare likninger. Det er også skrevet egne lærebøker som følger denne tette integrasjonen mellom programmering og fysikk (Langtangen, 2009; Malthe-Sørenssen, 2015). Malthe-Sørenssen et al. (2015, s. 7) opplever at en slik tilnærming er mer robust enn den tradisjonelle tilnærmingen, for med analytisk fysikk følger ikke utvalget av eksempler og regneoperasjonene nødvendigvis noen generell struktur. De er heller et resultat av prøving, feiling og erfaringer som er gjort gjennom årenes løp. Numerisk fysikk kan altså bidra til at elevene ser de store linjene, og ikke bare lærer seg å løse et mer eller mindre tilfeldig utvalg problemer.

2.5 Hvordan undervise algoritmisk tenkning?

Ifølge Futschek (2006, s. 160) er det like vanskelig å svare på hvordan man bør lære bort algoritmisk tenkning som det er å svare på hvordan man skal lære bort kreativitet. I dette delkapitlet har jeg likevel forsøkt å samle noen erfaringer som er gjort på området, fra forskningslitteraturen. Jeg har strukturert delkapitlet med formål om at det skal være praktisk anvendbart i klasserommet, som mer konkrete tips og forslag til undervisning med programmering.

Lav terskel, høyt tak

Sengupta, Kinnebrew, Basu, Biswas og Clark (2013, s. 362) peker på at et prinsipp for å inkludere algoritmisk tenkning i skolen, er at læringsaktivitetene må ha «en lav terskel og et høyt tak». Det må være lett for nybegynnere og enkelt å komme i gang, men samtidig ha mange muligheter og ikke gi begrensinger.

Som vi har sett i kapittel 2.3, kan programmering i seg selv være vanskelig for nybegynnere, og en må være varsom med å inkludere for mange vanskelige elementer på samme tid. Brown og Wilson (2018) anbefaler å holde seg til ett programmeringsspråk. Selv om hovedelementene ofte likner hverandre i ulike språk, vil det for en nybegynner være forvirrende å veksle mellom ulike syntakser.

Erfaringene til lærerne som har arbeidet med programmering i fysikk på Universitetet i Oslo, tilsier at programmering bør innføres separat før det benyttes som verktøy i fysikkundervisningen. Det kan bli «dobbelt så vanskelig» å lære både programmering og fysikk samtidig (Malthe-Sørenssen et al., 2015, s. 310). Samtidig er de opptatt av at programmering ikke bare skal være én økt i løpet av fysikk-kurset, men at studentene eksponeres for det jevnt

gjennom studiet. diSessa (2018, s. 14) valgte også å arbeide med programmering i to uker før elevene fikk begynne på selve matematikken og fysikken. Hans erfaring er imidlertid at sjetteklassingene ble utålmodige og syntes det ble for mye programmering før «det virkelig morsomme og interessante» begynte: anvendelsen av programmering i en naturfaglig kontekst. Hvis han skulle gjort det på nytt, ville han altså innført programmering mer gradvis på veien.

Læreren som modell og støtte

I sitt masterarbeid fant Sørby (2010, s. 179) at førsteårsstudentene på universitetet i fysikk ofte møter på problemer når de arbeider med oppgaver i numerisk fysikk, og han peker på at hjelp fra læreren kan være nødvendig. Samtidig så han at det i mange tilfeller var svært lite som skulle til for å lede studentene i riktig retning, og at studentene i samarbeid kan klare å finne riktig løsningsmetode.

Brown og Wilson (2018) har laget ti praktiske tips for hvordan man bør undervise programmering. Et av tipsene er at læreren bør skrive kode foran elevene, og ikke kun vise frem ferdig kode på en lysbildepresentasjon. Dette gjør at elevene får se hvordan en erfaren koder bygger opp programmet fra bunnen av, hvordan kode-editoren håndteres og hvilke snarveier som kan brukes, og hvordan man kan lete etter feil. Et annet viktig moment er at læreren vil holde tempoet nede og i større grad kunne åpne for spørsmål og elevdeltakelse. Det neste tipset innebærer nemlig at elevene bør få komme med hypoteser om hva de tror vil skje når programmet kjøres. Et problem med programmering, kan være at læreren helst skulle hjulpet mange elever samtidig. En mulig måte å håndtere dette på, er ifølge Brown og Wilson at læreren etter en felles gjennomgang stiller et diagnostisk flervalgsspørsmål, der elevene først avgir et svar, deretter diskuterer alternativene sammen i mindre grupper, for så å avgi et svar på nytt. Hvis misoppfatninger fortsatt forekommer, kan en åpne for nye klassediskusjoner.

Gode oppgaver og oppgavetekster

Futschek (2006, s. 160) sitt svar på å undervise algoritmisk tenkning, er å bruke velutviklede oppgaver, som ikke er for enkle å løse, men som er enkle å forstå. Begynneropplæringen kan dessuten være helt fri for faktisk programmering, der programmeringsspråkets syntaks kanskje er ukjent og forstyrrende for elevene. I stedet kan man bruke for eksempel pseudokode eller lage flytskjema. Selv om Futschek har skrevet spesifikt for informatikk, bør disse ideene kunne overføres til fysikkundervisningen.

Sørby (2010, s. 180) fant også at studentene ofte trenger hjelp med å legge en plan for å løse oppgavene. Han foreslår at selve oppgaveteksten kan gi denne hjelpen, hvis den er strukturert på en slik måte at studentene «bevisst eller ubevisst trenes i å lage en plan for modelleringsøkten» (Sørby, 2010, s. 188). Han mener at man i oppgaveteksten ikke må være redd for å forklare begreper og lære bort fysikkinnholdet. Formålet er ikke å teste elevene. Han finner at elevene fint klarer å oversette problemer til programmeringskode uten å forstå fysikkinnholdet som ligger bak. En dialogisk oppgavetekst kan altså hjelpe elevene med å knytte programmeringen til fysikken. Dette behøver imidlertid ikke bety at oppgavetekstene ikke kan inneholde mer åpne delspørsmål (Sørby, 2010, ss. 189-190).

Brown og Wilson (2018) anbefaler å bruke autentiske oppgaver, der elevene raskest mulig får prøve seg på oppgaver som vekker interesse, gjerne med bilder, lyd og video. De foreslår at elevene gjerne kan få noe ferdig utdelt kode, slik at de kan fokusere på en mindre og mer sentral oppgave. Til slutt anbefaler Brown og Wilson å ikke bare kode, men også arbeide med programmering på andre måter. Det kan være å bryte ned algoritmer i mindre deler, eller såkalte «Parsons-oppgaver» der elever får utdelt ferdige kodelinjer de selv må plassere i riktig rekkefølge.

En hybrid tilnærming

Orban, Teeling-Smith, Smith og Porter (2018) peker på ulike måter å tilnærme seg fysikk gjennom programmering. Den ene varianten, som de kaller «tradisjonell tilnærming» til programmering i fysikk, er at elevene programmerer stort sett alt på egen hånd. De setter initialbetingelser, gjør beregninger, og visualiserer eventuelt resultatene – gjerne med grafer. Den andre tilnærmingen er å bruke ferdige programmer, der elevene kun legger inn initialbetingelser og eventuelle tastaturkommandoer. Selve koden og beregningene som gjøres, er skjult for elevene, og de får umiddelbart visualisert resultatene. Vi snakker altså om en «svart boks», der elevene får simuleringer uten å måtte forstå beregningene som gjøres. Begge disse variantene har svakheter. Det kan være krevende for elever å lage animasjoner selv, så den første tilnærmingen begrenser gjerne hvilke visualiseringer som kan gjøres, eller at fysikken drukner i teknisk programmering. Den andre tilnærmingen kan gi avanserte visualiseringer, men elevene får ingen forståelse av hvordan beregningene utføres.

Orban et al. (2018) har derfor gått videre med en tredje, «hybrid tilnærming» til programmering. Her får elevene utføre det meste av beregningene selv, og resultatene sendes til en ferdiglaget animasjon. Slik kan elevene fokusere på de linjene i programmet som faktisk

handler om fysikk, mens den mer tekniske avanserte koden som driver selve simuleringen, er skjult for elevene. Dermed kan elevene arbeide med nokså avanserte visualiseringer, uten at den kognitive belastningen blir for stor. Kanskje kan en slik tilnærming også bidra til at elevene utvikler en forståelse for hvordan ulike komponenter kan utvikles av ulike personer og settes sammen til en større helhet – altså det Weintrop et al. (2016) kaller *systemtenkning* i sin taksonomi over algoritmisk tenkning. Hypotesen til Orban et al. (2018) er at en hybrid tilnærming, som gir en mer interaktiv og spill-liknende opplevelse, kan bidra til å øke motivasjonen for å programmere. Resultatene fra en elevundersøkelse støtter denne hypotesen, da de fleste elevene opplevde timene som ganske eller veldig morsomme. De fleste mente det var forholdsvis enkle utfordringer, men det var en del spredning i hvor mye elevene opplevde at programmeringsøkten bidro til økt forståelse for fysikkinnholdet (vektorer).

Sadiku et al. (2017, ss. 427-428) peker også på dilemmaet mellom å gi ferdige programmer eller å la studentene programmere selv. De peker på at ferdig materiale ofte er vanskelig å bruke og kan hindre læring, så de anbefaler at bruken begrenses til de tilfellene der det er nødvendig for å visualisere avanserte fysiske sammenhenger.

Sørge for godt samarbeid

Hvis mange elever står fast, kan det være utfordrende å være eneste lærer i klasserommet. Dette kan løses ved å la elevene arbeide i mindre grupper, samarbeide om koden og diskutere problemene med hverandre. Hutchins et al. (2018) studerte samarbeidslæring i en gruppe med videregående elever og erfarte at elevene hadde gode diskusjoner når de arbeidet sammen i gruppe om en programmeringsoppgave i fysikk. De ulike gruppene delte også refleksjoner med hverandre. Det viste seg imidlertid at enkelte personer i gruppa tok større ansvar enn andre, og før- og etter-testene viste også at noen elever hadde stort læringsutbytte av undervisningen, mens andre ikke hadde noen fremgang.

Brown og Wilson (2018) foreslår å la elevene arbeide sammen to og to på én datamaskin – der én skriver og én sitter ved siden av og bidrar med ideer og kommentarer. Rollene bør byttes flere ganger hver time. Roundy, Krebs, Schulte og Mulder (2015) har i et introduksjonskurs til programmering og beregningsorientert fysikk og matematikk, også fokusert på par-samarbeid og har gode erfaringer fra dette. De bytter også parene som samarbeider ofte, også midt i et prosjekt, slik at én av studentene må forholde seg til ny kode undervegs. De opplever at studentene blir mindre frustrerte og holder ut lenger, samarbeider bedre og skriver koden mer ryddig og forståelig, og at det frigjøres mer tid til læreren.

Kortemeyer og Kortemeyer (2018) har undersøkt hvordan studenter på universitetsnivå samarbeider om programmeringsoppgaver, og i hvilken grad de kopierer/plagierer hverandre når de løser prosjektoppgaver i fysikk. Studentenes oppgave var blant annet å studere bevegelsen til en ladet partikkel i et elektrisk og et magnetisk felt. Studentene fikk utdelt noe ferdig kode, som måtte tilpasses og utvides for å løse oppgavene. De finner at mange studenter har svært like besvarelser, og at en løsning gjerne videreformidles til mange studenter, uten at de selv er klar over at de nærmest skriver av hverandre. Dette er kanskje heller ikke uventet, gitt programmeringsspråkets strenge krav til struktur, og at programmering gjerne kan læres gradvis ved å sette sammen ulike elementer av ferdig kode. Bare et lite mindretall kopierer andres løsninger bevisst, og dette er i hovedsak studenter som tror de uansett ikke ville klart å løse oppgaven (Kortemeyer og Kortemeyer, 2018, ss. 16-17). Kortemeyer og Kortemeyer (2018, s. 17) foreslår at å inkludere koding på eksamen vil være en mulig løsning for å sikre at studentene setter seg godt nok inn i oppgavene og hindre at de kopierer hverandres kode. Det er da ikke snakk om å teste studentene i programmeringsspråkets syntaks, men deres ferdigheter i å utvikle algoritmer for å løse et fysikkproblem.

Lære å lete etter feil

Caballero, Kohlmyer og Schatz (2012) har kategorisert de ulike feilene studenter tidlig på universitetsnivå har gjort i en programmeringsoppgave i mekanikk. De finner at mange av feilene minner om feil som også gjøres med papir og blyant. Spesielt trekker de frem feil i beregning av krefter eller bevegelsesmengde. De finner også noen feil som er særegne for numerisk fysikk, og en del feil som ser ut som slurv. Et program kan kjøre fint dersom det ikke er noen syntaksfeil i koden, men det kan likevel være feil i beregningene og fysikken slik at resultatene blir gale.

En del studenter har problemer med å velge riktige startbetingelser i beregningene. Caballero et al. (2012, s. 8) har ikke klart å avgjøre om dette er slurvefeil etter å ha forandret på en tidligere versjon av programmet, eller om studentene ikke har klart å avgjøre hvilke verdier som er de riktige. Det viser likevel at det ikke er en triviell oppgave for studentene å identifisere og oppdatere variable.

Mange av studentene gjorde en eller flere feil i utregningen av krefter. Mange av feilene kan knyttes til manglende ferdigheter i vektorregning og innebærer blant annet feil i beregning av retning og absoluttverdi. Andre studenter inkluderte krefter som ikke var relevante for problemet i det hele tatt. Kanskje husket de bare kode fra tidligere, men var ikke i stand til å

avgjøre hva som var relevant for den nye situasjonen. En annen gruppe studenter plasserte beregningen av kraft utenfor en løkke i programmeringsspråket, slik at den ikke ble oppdatert hver gang. Slik regnet de som om kraften var konstant. Caballero et al. (2012, s. 9) gjør et poeng ut av at en slik feil sjelden ville forekommet i tradisjonell fysikk med papir og blyant, fordi en i disse tilfellene som regel alltid regner med konstante krefter.

Flertallet av studentene implementerte Newtons andre lov, på bevegelsesmengde-form, riktig. En vanlig feil var å oppdatere en bevegelsesmengde-vektor med en skalar kraft, eller å bruke en versjon av formelen som bare regnet ut endringen, uten å oppdatere bevegelsesmengden for hver iterasjon. Caballero et al. (2012, s. 9) tror at å kreve at studenter implementerer Newtons andre lov i programmene sine, kan bidra til å bevisstgjøre hvilke former av loven som kan være nyttige og anvendbare i ulike tilfeller.

Sørby og Angell (2012, s. 291) ser at flere studenter har problemer med å bestemme en passende verdi for tidsstegene i numerisk beregning med for eksempel Euler-metoden. Ofte settes «dt = 0.1» uten videre diskusjon. Dette er en verdi som gjerne fungerer godt, men i de tilfellene dette ikke er en god verdi, har studentene problemer med å finne ut hvor feilen ligger. Da studentene for eksempel fikk en uventet planetbane, begynte de å finne fysiske forklaringer på hvorfor banen hadde blitt som den ble – selv om det ikke er noe i modellen som kan forklare en slik oppførsel.

Selv om studentene har hatt introduksjon til programmering ved flere anledninger, finner Kortemeyer og Kortemeyer (2018, s. 16) at dette ikke har vært tilstrekkelig for at studentene skulle kunne arbeide selvstendig. Bare et mindretall studenter er komfortable med å skrive kode. Det viser seg at store deler av studentgruppen mangler oversikt over koden som helhet, og de løser oppgaven lite effektivt ved å sette sammen biter som et «lappeteppe». Dette passer godt med det Sørby og Angell (2012, s. 290) finner om at prøving og feiling er en vanlig brukt strategi blant studentene.

Caballero et al. (2012, s. 12) konkluderer med at det er viktig å ikke bare lære studentene å skrive riktig syntaks, men at de også får trening i å tolke feilmeldinger, finne og rette feil. Det gjelder også feil som ikke er direkte knyttet til programkoden – feil som gjelder kvalitativ tolkning av et fysisk problem og oversettelsen til et numerisk løsbart problem. Sørby og Angell (2012, s. 294) trekker en liknende slutning, om at studentene må bevisstgjøres forskjellene mellom modell og virkelighet, og om når de ulike verktøyene og kunnskapene kan anvendes.

Programmere for eksternt publikum

En mulig tilnærming til arbeid med programmering er å la elevene lage et produkt som er tiltenkt noen andre enn læreren og elevene selv. Pierson og Clark (2018) har i en undersøkelse blant sjetteklassinger sett på hvordan elevene lærer om tidevann, gravitasjon og treghet når de arbeider med datamodellering og blokkprogrammering. De gjennomførte det samme opplegget i to grupper; den første skulle lage modellen for et eksternt publikum (femteklassingene), mens den andre skulle lage den for egen klasse og læreren. Elevene fikk delvis ferdige modeller av sol, måne, jordklode og vann. Først ble elevene bedt om å vurdere og tilpasse modellen slik at den samsvarte med faktiske størrelsesforhold. Videre ble de bedt om å oversette kode for Jordas gravitasjon til menneskelig språk for å lære å lese kode, før de selv måtte lage tilsvarende kode for Månen og Sola. Over flere dager ble den digitale modellen gradvis utvidet etter hvert som elevene lærte ny fysikk. På den fjerde dagen fikk elevene tilbakemelding fra henholdsvis noen femteklassinger og noen medelever, som de brukte til å gjøre tilpassinger av modellene. Den femte dagen ble den første gruppen bedt om å skrive en «brukermanual» for programmet sitt, mens den ande gruppen ble bedt om å skrive en «rapport». Studien konkluderer med at elevene som programmerte for et eksternt publikum, hadde større læringsutbytte enn elevene som kun programmerte for læreren. Den første gruppen la ned større arbeid for å gjøre fysikken forståelig for femteklassingene, viste større engasjement og hadde rikere diskusjoner i klassen.

Det kan dermed se ut til at elever motiveres av å arbeide med reelle problemstillinger som også er nyttige for andre enn dem selv. Å lage oppgaver eller prosjekter der elever programmerer for eksterne publikum, kan være en måte å inkludere ingeniørvitenskapelige praksiser og algoritmisk tenkning i fysikkfaget. Eksempelet med tidevann viser også at dette kan gjøres med nokså teoretiske fenomener.

Eulers metode og en tikkende klokke

diSessa (2018, ss. 9-10) bruker en «tikke-modell» for å gi sjetteklassingene en intuitiv tilnærming til Eulers metode for numeriske beregninger (se kapittel 2.6). Modellen går ut på at visse beregninger utføres hver gang klokken «tikker». I hans eksempel, med en fallende ball, vil det si at ny fart og posisjon beregnes ved hvert tikk, og at ballen flyttes i henhold til dette. Dette er altså en variant av det som i programmering kalles «løkker», gjerne kjent som «for» eller «while» (se f.eks. Savitch, 2013, s. 95). diSessa (2018, ss. 10-11) hevder tikke-modellen er enkel å forstå for sjetteklassinger, og at man med en slik modell lettere kan gi forståelse for hvordan kraft og akselerasjon bidrar til bevegelse. Videre peker han på fordeler som at tikke-

modellen er høyst generaliserbar til å beskrive alle typer bevegelse, at den er konseptuelt forståelig, selv om man ikke kjører programmet, og ikke minst at den oppleves som morsom blant elevene. Elevene utvidet programmene sine og lagde egne spill, og en slik allmenn tilknytning er ifølge diSessa «et fantastisk prinsipp for læring» (diSessa, 2018, s. 11, min oversettelse).

Mens diSessa går fra det abstrakte programmeringsspråket til et visuelt resultat, valgte Yaşar (2013) en mer induktiv metode da han underviste noe eldre studenter i den samme typen numeriske beregninger, av typen han kaller «ny = gammel + endring». Han valgte en tilnærming der studentene først undersøkte en bevegelse, for eksempel et svingende lodd i en fjær, i et digitalt simuleringsprogram (Interactive Physics). Senere skulle de bruke regneark (Excel) og prinsippene for numerisk integrasjon til å beregne den samme bevegelsen. Dette gir en god oversikt over hva som skjer i hvert steg, men har sine begrensinger i at det ikke er mulig å gjøre veldig lange og avanserte beregninger. Derfor får Yaşar studentene til å innse at det er fordelaktig å implementere beregningene i et programmeringsspråk (Python). Yaşar (2013, s. 17) peker på at å utføre den samme modelleringen på flere måter, kan hjelpe studentene i å se fordeler med ulike verktøy og forstå hvorfor de bør lære programmering. I en evaluering av kurset svarte 90 prosent av studentene at de mente modellering bidro til økt forståelse av prinsipper i naturvitenskap (Yaşar, 2013, s. 16).

Eksemplene over viser at det er mulig å forenkle teorien og matematikken ved numeriske beregninger til et minimum, men samtidig beholde styrken som gjør det praktisk anvendbart i mange fysikk-sammenhenger. Yaşars opplegg viser dessuten at det ikke er nødvendig å ta i bruk programmeringsspråk med en gang, og at regneark kan være en mulig inngang til programmering.

Integrere programmering på fagets premisser

En av lærerne som ble intervjuet i FYS21-prosjektet (Angell, Kind og Henriksen, 2008) brukte digital programvare til å gjøre modellering og simulering i fysikkundervisningen. Han hadde imidlertid mindre gode erfaringer med dette, da elevene ikke så ut til å vise særlig interesse for arbeidet. Lærerens opplevelse var at arbeidet som ble gjort på datasalen var tidkrevende og at det av elevene ble sett på som noe man gjorde ved siden av det «egentlige faget» (Angell, Kind og Henriksen, 2008, s. 118). Det har riktig nok skjedd en del med teknologien siden den gang – blant annet er begrepet «datasal» så godt som borte, og elevene har rask tilgang til egne bærbare datamaskiner i klasserommet. Datamaskinen er nok i dag også i mye større grad

integrert i matematikk- og fysikkundervisningen. Elevene er for eksempel vant til å bruke GeoGebra, CAS og regneark som verktøy i undervisningen. Det kan nok bidra til at også digital modellering vil fungere bedre i dag. Likevel bør en være oppmerksom på at programmering fort kan oppleves tilsvarende – som noe som gjøres i tillegg til den «egentlige» fysikken. Vi har sett hvordan elevene til Caballero et al. (2013, s. 41) måtte lære seg programmeringsspråket på nytt, fordi det gikk for lang tid mellom hver gang de brukte det i undervisningen. Utfordringen blir å integrere programmering og algoritmisk tenkning på en god måte i de øvrige emnene i læreplanen slik at elevene opplever det både som et nyttig verktøy og at det kan bidra til økt forståelse av fysikk-fenomener.

Weintrop et al. (2016) har utviklet en del undervisningsopplegg for videregående skole basert på deres rammeverk for algoritmisk tenkning i naturfag. Opplegget de presenterer for fysikk, handler om å bruke en skjermopptaker til å filme når elevene spiller et spill, for eksempel Angry Birds. Deretter kan de analysere filmen med et analyse- og modelleringsverktøy (Physics Tracker) og bruke dette til å bestemme for eksempel tyngdeakselerasjonen i spillverdenen eller undersøke om energi er bevart. Noen av de praksisene som inngår i et slikt undervisningsopplegg, vil være innsamling, visualisering og analysering av data, bruk av modeller til å finne og teste løsninger, vurdere modellene og å kommunisere informasjon om et system (Weintrop et al., 2016, s. 142). Dette opplegget kunne imidlertid vært innført i norsk fysikkundervisning i dag, og det anvender ikke programmering eller algoritmer i særlig grad. Det gjør derimot opplegget de har laget for biologi. Der blir elevene først bedt om å løse en oppgave (sette tegn i et passord i rekkefølge), før de så skal lage en algoritme og skrive en pseudokode for hvordan en datamaskin skulle løst oppgaven. Deretter blir elevene bedt om å følge hverandres algoritmer, og til slutt reflektere over hvorfor datamaskinen er et viktig verktøy i beregning av DNA-struktur. Slik lærer elevene å strukturere programkode og utvikle algoritmer, og de får forståelse for hvordan beregninger kan være nyttig i faglig sammenheng, helt uten faktisk å programmere på en datamaskin.

KURT – Kompetansesenter for undervisning i realfag og teknologi (2019) ved Universitetet i Oslo holder etterutdanningskurs i programmering for realfagslærere. Tellefsen (2018) gir flere eksempler på hvordan realfaglig programmering kan benyttes for fagenes skyld. Et av eksemplene er fra fysikk og numerisk beregning av fall med luftmotstand. Slik kan altså modelleringen av de fallende muffinsformene fra FYS21-prosjektet (Angell, Kind, Henriksen, et al., 2008) videreføres med en programmeringsdel. Hun gir også eksempel på at elevene kan bruke programmering til å lage sine egne kalkulatorer, og bruker andregradsformelen fra

matematikk som eksempel. Her må elevene lage seg en algoritme som tar hensyn til mulige tilfeller, for eksempel fortegnet til diskriminanten under rottegnet, for å gi ut riktig svar. Tilsvarende formelkalkulatorer bør også kunne lages for fysikkformler.

Vi ser at fallende gjenstander er en gjenganger i litteraturen. Både Tellefsen (2018), diSessa (2018), Sherin (2001) og Malthe-Sørenssen et al. (2015) trekker frem dette som et godt eksempel på hvorfor integrere beregninger i fysikkundervisningen, og også Weintrop et al. (2016) bruker fallende «Angry Birds» i deres fysikk-opplegg. Dette er helt klart et godt utgangspunkt for hvordan elever i videregående skole kan arbeide med programmering og numeriske beregninger. Det er en rettlinjet bevegelse, med et begrenset antall krefter og parametere å ta hensyn til, og det blir en forholdsvis oversiktlig beregning som passer nivået på videregående skole. Dette bør imidlertid ikke være det eneste argumentet for at elevene skal programmere i fysikk. Det finnes nødvendigvis en rekke andre problemstillinger som lar seg løse numerisk, men utfordringen er at det fort kan bli så avansert at det ikke er egnet for fysikkelever som lærer programmering for første gang. De fleste bevegelser er to- og tre-dimensjonale, og slike bevegelser, hvor man må håndtere flere kraftkomponenter i ulike retninger, er et betydelig steg opp i vanskelighetsgrad. Vi ser også at de fleste eksemplene inneholder en form for krefter og bevegelse, og det er vanskelig å finne gode eksempler innenfor tema som ikke innebærer mekanikk. I kvantefysikk kan for eksempel Schrödingerlikningen og energinivåer løses numerisk, men det er langt utenfor dagens kvalitative introduksjon til moderne fysikk på videregående nivå. I termisk og statistisk fysikk kan sannsynlighetsmodeller brukes til for eksempel å beskrive en partikkels virrevandring, men heller ikke dette er særlig interessant for elever på videregående skole som følger dagens læreplan.

2.6 Kort introduksjon til numeriske beregninger

Prinsippet bak numeriske beregninger er å gi en tilnærmet løsning av et matematisk problem ved å bruke tall og algoritmer. Ettersom numeriske løsningsmetoder gir en naturlig anvendelse av programmering i fysikk og er en sentral del av undervisningsoppleggene som presenteres i kapittel 4, gir jeg her en noe mer faglig introduksjon til noen enkle numeriske metoder. Dette er inkludert her for å gi leseren en oversikt over sentrale begreper og metoder som kan være aktuelle for fysikkelever i videregående skole.

Det er særlig hensiktsmessig å bruke numeriske løsningsmetoder på problemer som vanskelig lar seg løse eksakt. En andregradslikning kan enkelt løses eksakt med den velkjente andregradsformelen, men selv utledningen av denne kan være nokså avansert for elever, og det

blir raskt vanskeligere å løse likninger av høyere grad. Det finnes også (avanserte) formler for generelle tredje- og fjerdegradslikninger, men i 1824 beviste Niels Henrik Abel at det ikke engang finnes noen slik generell formel for femtegradslikninger (Lindstrøm, 2006, ss. 148, 485). Derimot er det ofte fullt mulig å finne gode tilnærmede verdier av løsningen med standardiserte numeriske løsningsmetoder. Det finnes en rekke både enkle og mer avanserte metoder for å gjøre numeriske beregninger (se for eksempel Quarteroni, Sacco og Saleri, 2007). De enkleste metodene er relativt intuitive og raske å implementere, men kan til gjengjeld ofte være mindre effektive, ha begrenset gyldighetsområde og gi større feil. Mer avanserte metoder vil kunne gi en betydelig bedre tilnærming, men er gjerne mindre selvforklarende og vanskeligere å implementere.

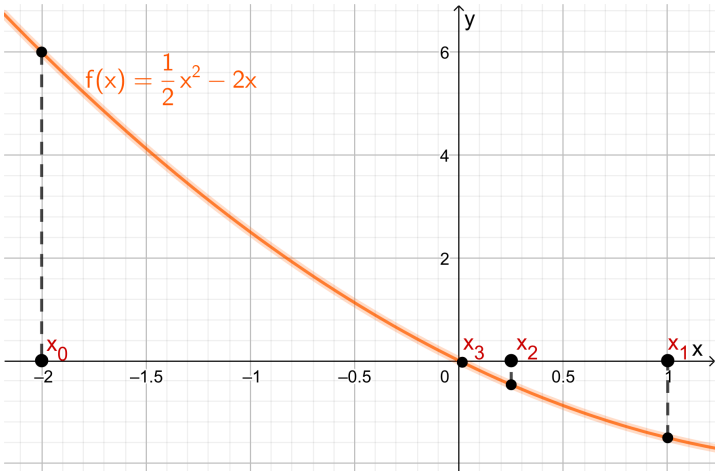
Ut fra de siste læreplanskissene og læreplanene som er sendt til høring for matematikk på videregående skole (Utdanningsdirektoratet, 2019c), ser det ut til at elevene skal lære å løse likninger numerisk, utføre numerisk derivasjon og utføre numerisk integrasjon. Jeg gir her eksempler på noen enkle metoder som kan være aktuelle i skolen, og som vil være aktuelle å anvende for å løse problemer numerisk i fysikk. Det finnes mye teori om hvor metodene er egnet og hvorvidt de gir konvergente løsninger, som kan være aktuelt i læreplanen for matematikk, men jeg velger å ikke inkludere dette her. For detaljer henvises leseren til en tekst om numerisk matematikk, for eksempel Quarteroni et al. (2007), Kreyzig (2011) eller Lindstrøm (2006).

Å finne nullpunkter numerisk

En likning på formen $f(x)=0$ kan ofte, med noe algebra, enkelt skrives på formen $x=g(x)$. Ved å sette inn en vilkårlig verdi x_0 i funksjonen på høyre side, vil vi få en ny verdi $x_1=g(x_0)$, som under visse betingelser er en litt bedre tilnærming av nullpunktet. Ved å gjenta operasjonen vil løsningen bli en stadig bedre tilnærming. Denne metoden kalles *fikspunktiterasjon*.

$$x_{n+1}=g(x_n) \quad \text{for } n=0,1,2,\ldots$$

Figur 2-4 viser hvordan løsningen med fikspunktmetoden etter få steg nærmer seg det faktiske nullpunktet $x=0$ til funksjonen $f(x)=\frac{1}{2}x^2-2x$. Likningen kan skrives på formen $x=g(x)=\frac{1}{4}x^2$. Med $x_0=-2$ blir $x_1=1$, $x_2=0{,}25$ og $x_3=0{,}02$.



Figur 2-4 Eksempel på bruk av fikspunktiterasjon.

Det finnes andre metoder for å løse likninger numerisk som er både mer effektive og mer presise enn fikspunktmetoden. Noen kjente er for eksempel *halveringsmetoden* og *Newtons metode*.

Å finne den deriverte numerisk

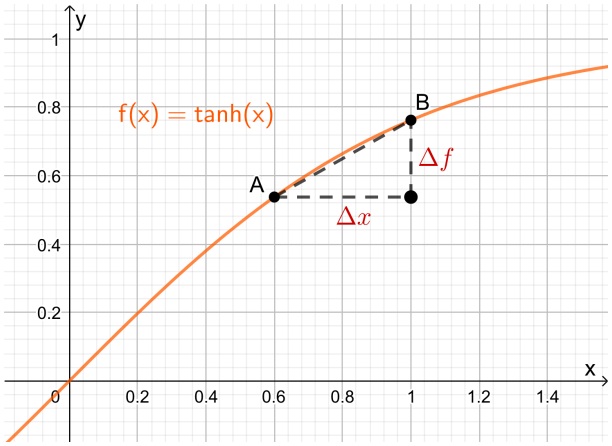
Den matematiske definisjonen av den deriverte av en funksjon f i punktet x er

$$f'(x) = \lim_{\Delta x \to 0} \frac{\Delta f}{\Delta x} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

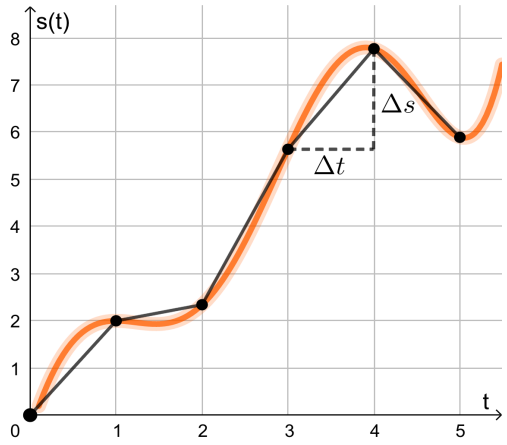
Grenseverdien kan tilnærmes numerisk ved å la Δx bli svært liten, uten at den blir helt null. Det gir en numerisk tilnærming av den deriverte:

$$f'(x) \approx \frac{\Delta f}{\Delta x}$$

Figur 2-5 viser hvordan den deriverte av funksjonen $f(x) = \tanh(x)$ kan estimeres for $x = 0{,}6$. Her velges $\Delta x = 0{,}4$, som gir $\Delta f = 0{,}22$, og dermed er $f'(0{,}6) \approx 0{,}22 / 0{,}4 = \underline{0{,}55}$. Det avviker noe fra den faktiske verdien på omtrent 0,71. Hvis Δx gjøres mindre, blir tilnærmingen stadig bedre. Ved å gjenta for flere punkter på grafen, kan vi beregne kurven til den deriverte for flere x-verdier.



Figur 2-5 Eksempel på numerisk derivasjon.



Figur 2-6 Beregne fart numerisk fra posisjonsdata.

Figur 2-6 viser hvordan numerisk derivasjon kan brukes i en fysikksammenheng. Her er det gjort posisjonsmålinger hvert sekund (svarte punkter) for en fysisk bevegelse (oransje kurve). Da kan vi estimere farten i hvert målepunkt numerisk ved å bruke numerisk derivasjon mellom det aktuelle punktet og det påfølgende punktet:

$$v(t_i) = s'(t_i) \approx \frac{s(t_{i+1}) - s(t_i)}{t_{i+1} - t_i}$$

Denne teknikken benyttes i undervisningsopplegg 1 i kapittel 4.3.

Å finne den integrerte numerisk

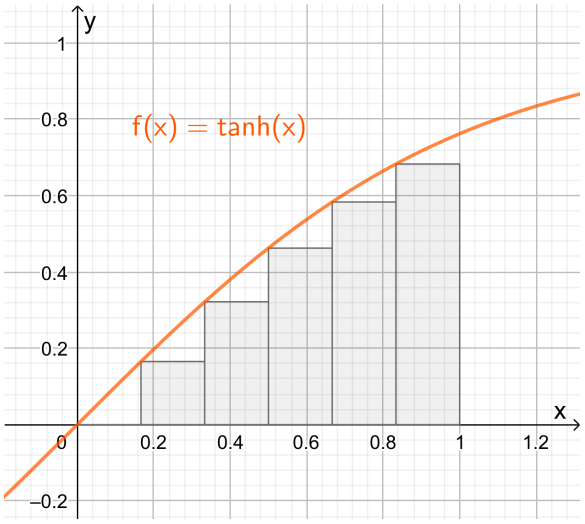
Hvis vi lar $F(x)$ være en funksjon slik at $F'(x) = f(x)$, har vi fra metoden for numerisk derivasjon at

$$f(x) = \frac{F(x + \Delta x) - F(x)}{\Delta x}$$

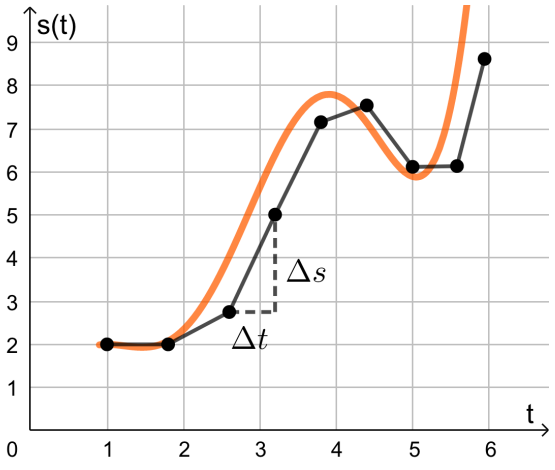
som gir

$$F(x + \Delta x) = F(x) + f(x) \cdot \Delta x$$

Da kan vi finne verdien til den integrerte funksjonen av f hvis vi kjenner verdien $F(x_0)$ i et startpunkt. Dette er *Eulers metode*, som blant annet Yaşar (2013) pedagogisk kaller «ny = gammel + endring» og som diSessa (2018) beskriver med en «tikke-modell».



Figur 2-7 Eksempel på numerisk integrasjon.



Figur 2-8 Numerisk beregning av posisjon fra fartsdata.

Figur 2-7 viser hvordan Eulers metode kan brukes til å finne det bestemte integralet til funksjonen $f(x) = \tanh(x)$ for x mellom 0 og 1. Her er $\Delta x = 0{,}17$ og Euler-metoden gjentas 6 ganger for å gi en tilnærmet verdi av arealet under grafen.

I fysikk kan vi for eksempel bruke eulermetoden til å finne posisjonen etter et lite tidssteg Δt hvis vi kjenner farten $v(t)$ og posisjonen ved start: $s_1 = s_0 + v(t) \cdot \Delta t$. Når denne operasjonen gjentas flere ganger, kan vi regne ut posisjonsdataene for hele bevegelsen. Dette er illustrert i Figur 2-8. Vi ser at den numerisk beregnede posisjonsgrafen (svart kurve) avviker noe fra den faktiske posisjonsgrafen (oransje kurve). Med mindre tidssteg Δt, ville disse to grafene sammenfalt i større grad. Denne teknikken er særlig nyttig for å beregne fart og posisjon når kreftene/akselerasjonen er kjent:

$$v(t_{i+1}) = v(t_i) + a(t_i) \cdot \Delta t$$

$$s(t_{i+1}) = s(t_i) + v(t_i) \cdot \Delta t$$

Denne teknikken benyttes i undervisningsopplegg 2 i kapittel 4.3.

2.7 Oppsummering

Litteraturstudien som er presentert i dette kapitlet, viser at det foreløpig er gjort begrenset med empirisk forskning på programmering og algoritmisk tenkning i fysikk på videregående nivå, men at det er et svært aktuelt forskningsområde hvor det stadig presenteres nye publikasjoner.

Begrepet «algoritmisk tenkning» blir brukt som en direkte norsk oversettelse av «computational thinking» – en problemløsningsstrategi og tenkemåte som baserer seg på å bruke sentrale prinsipper i informatikk og datavitenskap. Sentrale sider ved algoritmisk tenkning er å samle, håndtere og visualisere data, modellere, simulere, generalisere og abstrahere, bryte ned større problemer i mindre, mer løsbare delproblemer, og å se systemer som en helhet. Det er mange store visjoner om hvordan programmering og algoritmisk tenkning kan bidra til økt læring og være en viktig problemløsningsstrategi. Til nå har imidlertid forskningen brukt mye tid på å definere «computational thinking» som begrep og hva det innebærer for undervisningen, og det er lite forskning som undersøker hvordan denne tenkemåten påvirker læringsutbyttet og forståelsen i fysikk.

Det kan være ulike hensikter med å innføre programmering som en del av fysikkundervisningen. Det kan argumenteres for at programmering er et såpass viktig verktøy for fysikere og ingeniører at elevene bør lære seg å bruke dette verktøyet tidlig, samtidig som det gir et riktigere bilde av hvordan fysikk anvendes i forskning og arbeidslivet. Det er også muligheter for at programmering kan gi alternative tilnærminger til det som tradisjonelt har vært innholdet i fysikkfaget, og at dette kan gi en dypere forståelse for begreper og fenomener. Det finnes noe forskning som kan antyde at programmering kan bidra til kunnskapsintegrerende prosesser og konseptuell fysikkforståelse (Taub et al., 2015). Det er også funn som kan tyde på at programmering kan være godt egnet til å beskrive kausalitet og årsak–virkning (Sherin, 2001). Det pekes på hvordan programmering kan brukes til å bygge opp fysikkfaget på andre måter ved å anvende noen få, enkle numeriske teknikker (Malthe-Sørenssen et al., 2015). Det kan bidra til å fokusere på de sentrale fysiske prinsippene, unngå urealistiske antakelser og vanskelig matematikk. Det ser også ut til at programmering kan ha en nyttig rolle for modellering, samt å være grunnlag for diskusjoner mellom elevene (Caballero et al., 2013). Samtidig er det også tegn på at programmering kan bidra til kognitiv overbelastning og at elevene arbeider i adskilte moduser (Sørby og Angell, 2012; Taub et al., 2015).

Basert på litteraturen kan vi trekke ut noen erfaringer om hvordan programmering kan integreres i praksis. Det pekes på at det bør utformes gode oppgaver og oppgavetekster (Futschek, 2006; Sørby, 2010) med «lav terskel og høyt tak» (Sengupta et al., 2013), gjerne

som en «hybrid tilnærming» mellom at elevene skriver kode selv og at de får noen ferdige komponenter (Orban et al., 2018). Det er mulig at engasjementet og læringsutbyttet øker når elevene får lage programmer for et eksternt publikum. Læreren har en viktig rolle som modell og støtte. Ved å sørge for godt elevsamarbeid oppstår gode diskusjoner mellom elevene, de hjelper hverandre og det frigjøres tid som læreren kan bruke til å hjelpe flere (Brown og Wilson, 2018; Hutchins et al., 2018; Roundy et al., 2015; Sørby, 2010).

Jeg har presentert noen enkle numeriske teknikker som er særlig egnet i fysikk. Spesielt numerisk derivasjon og integrasjon er nyttig for å regne med kraft og bevegelse. Eulers metode blir i litteraturen beskrevet med en «tikke»-modell (diSessa, 2018) og som «ny = gammel + endring» (Yaşar, 2013).

3 INTERVJU-UNDERSØKELSE: Fysikklæreres syn på programmering

Programmering og algoritmisk tenkning skal inn i skolen, men vet lærerne hva dette innebærer, og har de noen formening om hvordan dette kan bidra til læring i fysikk? Hvilke utfordringer ser de? Ettersom algoritmisk tenkning og programmering i liten grad eksisterer i skolen i dag, og det fremdeles finnes få erfaringer internasjonalt, er det interessant å finne ut hva noen norske fysikklærere ser for seg at det kan være. Høringsuttalelsene til kjerneelementene viser at programmering er noe som opptar mange, og en stor andel er skeptiske til innføring av teknologi og programmering i matematikk- og naturfagene (Utdanningsdirektoratet, 2017a, 2017b). I en pilotstudie for masterarbeidet, gjennomførte jeg høsten 2018 intervjuer med tre fysikklærere ved en videregående skole i Norge for å undersøke om en tilsvarende skepsis er representert blant fysikklærerne, hvilke muligheter og utfordringer de ser programmering kan gi for læring av fysikk, og hvilken forståelse av algoritmisk tenkning som ligger til grunn. Dette kapitlet er i stor grad hentet fra pilotarbeidet og er tidligere levert som en eksamensbesvarelse i et emne om forskningsmetoder. Det gjengis her i en delvis omskrevet og tilpasset versjon for leserens oversikt og arbeidets helhet.

Dette kapitlet svarer på følgende forskningsspørsmål:

2. Hvordan forstår fysikklærere begrepet «algoritmisk tenkning», og hvilke utfordringer og muligheter ser lærerne for programmering i fysikkundervisningen?

Jeg tar først for meg forskningsmetoden som er brukt, utvalg, metodologi og noen etiske betraktninger i kapittel 3.1. Jeg presenterer funnene i kapittel 3.2 og diskuterer implikasjonene vi kan trekke fra dem i kapittel 3.3.

3.1 Metode

Ettersom programmering fremdeles ikke er innført i fysikkfaget og lærerne ikke underviser i programmering og algoritmisk tenkning i dag, er det vanskelig å undersøke forskningsspørsmålet ved å observere lærernes praksis. Det ble derfor ganske tidlig klart at det mest hensiktsmessige var å spørre lærerne selv, og intervju stod frem som en godt egnet metode. Et viktig formål ved intervjuene var å avklare lærernes forståelse av begrepet «algoritmisk tenkning», og intervjuene har dermed elementer av et begrepsintervju (Kvale og Brinkmann, 2015, s. 180).

Det finnes en rekke ulike epistemologiske og metodologiske tilnærminger til hva slags kunnskap vi kan skaffe og hvordan vi kan skaffe den. Jeg følger her ideen etter Thomas Kuhn (1970, ss. 10-11) om at de forskningsmetodene vi bruker og den kunnskapen vi frembringer, er basert på det forskningsparadigmet og den tradisjonen vi tilhører. Jeg følger en pragmatisk middelvei mellom filosofisk dogmatisme og skeptisisme, og at kunnskap både er konstruert og basert på den virkeligheten vi opplever (Robson og McCartan, 2016, ss. 28-29). Undersøkelsen tilhører det kvalitative paradigmet. I motsetning til en kvantitativ undersøkelse, gir dette mer datamateriale om få tilfeller og gjør det dermed mulig å studere disse tilfellene grundigere. Kvalitativ forskning egner seg til å studere enkeltindivider sett fra individets perspektiv (Robson og McCartan, 2016, ss. 18-20).

Utvalg

Intervjuene ble gjennomført med tre fysikklærere, som utgjør utvalget for datainnsamlingen. De tre lærerne, som her har fått navnene Anders, Birgitte og Camilla, er alle ansatt ved den samme videregående skolen i Norge og er av praktiske årsaker rekruttert via eget nettverk. De tre lærerne ble alle spurt om de ønsket å delta i et intervju om deres tanker rundt programmering i fysikkundervisningen i forbindelse med læreplanfornyelsen. Samtlige svarte positivt.

De tre lærerne underviser i fysikk på det som må beskrives som en nokså typisk, men moderne, videregående skole med relativt god søking i utkanten av en norsk storby. Lærerne er høyt utdannet, med både mastergrader og doktorgrader i fysikk ved universiteter i både inn- og utland, og flere har også erfaring med fysikk fra arbeidsliv utenfor skolen. Anders og Camilla har undervist i fysikk ved den aktuelle skolen i flere år. Camilla har også undervisningserfaring fra universiteter i Norge og utlandet. Birgitte har tidligere arbeidet flere år som sivilingeniør og som naturfaglærer i ungdomsskolen, og hun er nå ansatt i et vikariat. På tidspunktet for intervjuet hadde hun derfor ikke like mye erfaring med fysikkundervisning som de andre to.

Alle lærerne oppgir at de har erfaring med programmering, hovedsakelig fra egen utdanning og tidligere arbeidserfaring. Anders og Camilla har også undervist faget «Teknologi og forskningslære» i flere år, der de har fått noe erfaring i å undervise programmering. Camilla har dessuten forsøkt å introdusere elevene sine for enkel programmering når hun har undervist matematikk, selv om dette ikke står eksplisitt i dagens læreplaner. Birgitte har aldri undervist i programmering. Anders tar nå et kurs for lærere om programmering i skolen. Camilla er den eneste læreren som hadde lest de første utkastene til nye læreplaner i matematikk da intervjuet ble avholdt.

At de tre lærerne er rekruttert blant de nærmeste og mest praktisk tilgjengelige, gjør at utvalget må beskrives som et lite bekvemmelighetsutvalg. Det kan være mindre tilfredsstillende, ettersom resultatene ikke nødvendigvis er representative (Robson og McCartan, 2016, ss. 280-281). At lærerne er ansatte ved den samme skolen, kan for eksempel medføre at de er mer samstemte i sitt syn på programmering eller fysikkundervisning generelt. Likevel mener jeg disse lærerne utgjør et relevant utvalg. Lærerne representerer et kollegium som eksisterer ved det som kunne vært en vilkårlig norsk videregående skole, og skiller seg ikke vesentlig fra hva som finnes andre steder. Lærerne er godt kvalifiserte, men de har ikke nødvendigvis noe mer erfaring med programmering enn det vi kan anta at fysikklærere generelt har. At Birgitte har vært ansatt ved skolen i kortere tid, kan også medføre at hun bidrar med et noe annet perspektiv. Selv om funnene altså ikke er statistisk generaliserbare (Robson og McCartan, 2016, s. 166), er det sannsynlig at disse lærernes synspunkter også forekommer blant andre lærere, og slik har funnene en overføringsverdi og kan gi muligheter for analytisk generalisering (Kvale og Brinkmann, 2015, s. 291).

Utover at intervjuet skulle handle om fysikk og programmering, fikk lærerne ingen informasjon om selve innholdet i intervjuet. Jeg ønsket ikke at lærerne skulle sette seg inn i temaet og begreper spesielt for intervjuet. På den måten er de kanskje mer representative for dagens fysikklærere, enn hvis de hadde fått forberede seg mer. En utfordring ved dette kan være at lærernes svar blir mindre utdypende og gjennomtenkte, og det kan også være lærerne opplever at de ikke kan svare på spørsmålene. Et viktig etisk prinsipp i forskningen er å beskytte menneskeverdet til deltakerne (NESH, 2016, s. 12), og for å redusere belastningen av lærernes integritet, var jeg derfor påpasselig med å informere om at jeg bare vare ute etter lærernes synspunkter, og at det ikke var noen riktige eller gale svar. Før intervjuet ble alle deltakerne presentert for formålet med undersøkelsen, hva deltakelsen innebærer, konfidensialitet,

håndtering av personopplysninger, og retten til å trekke seg, i henhold til de etiske retningslinjene (NESH, 2016) og NTNUs retningslinjer for håndtering av personvern (NTNU, 2018). Samtykke til innsamling av data ble innhentet ved en skriftlig samtykkeerklæring (se vedlegg B). Prosjektet er meldt til og godkjent av Norsk senter for forskningsdata (NSD).

Forløp

Lærerne ble intervjuet individuelt. Det ble delvis gjort av praktiske årsaker, og delvis for at læreren skulle få uttrykke sine egne syn på programmering uten at kollegaene var til stede og eventuelt påvirket utsagnene. Det var jeg personlig som gjennomførte intervjuene i et møterom på lærernes arbeidsplass. Samtalene ble spilt inn med en digital lydopptaker, og hver samtale hadde en varighet på i underkant av en halv time.

Intervjuene kan klassifiseres som semistrukturerte ved at de hadde en viss planlagt struktur med noen spørsmål og tema som skulle dekkes, men nokså stor frihet i rekkefølge, spørsmålsformulering, tid og oppfølging av deltakernes svar (Robson og McCartan, 2016, ss. 290-291). Detaljene i intervjuet ligger i den vedlagte intervjuguiden (vedlegg A).

I den første delen av intervjuet ble deltakerne stilt spørsmål om deres erfaring med programmering, algoritmisk tenkning og hvordan de ser for seg at dette kan inkluderes i fysikkundervisningen. På den måten snakket de om programmering og algoritmisk tenkning fra flere ulike perspektiver. Først til sist i intervjuet ble de fire hovedkategoriene i det teoretiske rammeverket for algoritmisk tenkning (Weintrop et al., 2016, se kapittel 2.2) presentert for deltakerne, og de ble bedt om å fortelle hva de tenker disse innebærer i fysikkundervisningen. Dette ble gjort for å se hva lærerne i utgangspunktet forbinder med programmering og algoritmisk tenkning, og for å sikre at alle sider ved rammeverket er berørt.

Samtalene gikk relativt lett. Lærerne snakket velvillig, men det var behov for enkelte oppfølgingsspørsmål, og undersøkelsen bar kanskje noe preg av at lærerne ikke hadde tenkt noe særlig på programmering i fysikkundervisning fra før. Lærerne var tydelig noe mer usikre og kortfattet når de fikk spørsmål om mindre kjente begreper knyttet til algoritmisk tenkning.

Validitet

I undersøkelsen er intervju benyttet som metode, og følgelig er undersøkelsen inspirert av sosial-konstruktivisme. Kunnskapen konstrueres i samtalen mellom deltakeren og forskeren i intervjuet, og meninger blir til ved at mennesker samhandler med hverandre (Robson og McCartan, 2016, ss. 24-25). Det medfører nødvendigvis at forskeren tar aktivt del i

undersøkelsen og kan slik påvirke resultatene. Forskeren vil bringe sine egne verdier og overbevisninger inn i undersøkelsen, og slik er det en fare for forventningsskjevhet. Betydningen av dette kan imidlertid reduseres dersom forskeren er klar over sine verdier og er åpen for motstridende funn (Husén, 1988, s. 19; Robson og McCartan, 2016, ss. 148-149). Det er altså en mulighet for at jeg som intervjuer, ubevisst kan ha påvirket holdningen til intervjupersonene i for eksempel en positiv eller negativ retning. På tidspunktet for intervjuet oppfattet jeg imidlertid meg selv som ganske nøytral i spørsmålet om programmering i skolen, og som det fremgår av intervjuguiden (vedlegg A) er de fleste spørsmålene formulert ganske nøytrale, eller etterfulgt av et tilsvarende spørsmål med motsatt ladning. Et unntak er kanskje spørsmålet «Kan det [programmering og algoritmisk tenkning] bidra til økt læring i fysikk?», som kan være noe ladet. Her kan det være lett for læreren å svare «ja». Dette spørsmålet kunne muligens vært noe bedre formulert, og som også erfaringer fra blant annet vitneavhør viser, kan små variasjoner i spørsmålsstillingen påvirke svarene betydelig (Kvale og Brinkmann, 2015, s. 180). Ettersom intervjuene var semistrukturerte, vil jeg bemerke at spørsmålene kan ha blitt formulert annerledes, og i en annen rekkefølge, basert på samtalens forløp og hva læreren selv trakk frem. Resultatene viser eksempler på at lærerne også kom med kritiske kommentarer og viste skepsis.

Kvale og Brinkmann (2015, s. 114) peker på at kvalitativ forskning ofte forstås ut fra sin kontekst, og de peker på makrososiologiske kontekster som kultur og samfunn, eller mikrososiologiske kontekster, som miljø og situasjon. At intervjuene foregikk på lærernes egen arbeidsplass, kan ha bidratt til at miljøet oppleves kjent og trygt. Samtidig kan det også ha påvirket lærerne til for eksempel å snakke mer positivt om programmering, fordi de ønsker å representere skolen på en god måte, slik at den fremstår fremoverlent og moderne. At lærerne ble intervjuet individuelt uten andre kollegaer til stede, og med informasjon om intervjuerens taushetsplikt, kan imidlertid ha bidratt til å redusere dette. At det ble gjennomført tre intervju, bidrar også til triangulering og kan bidra til å styrke undersøkelsen validitet (Robson og McCartan, 2016, s. 171).

Intervjuet utgjør i seg selv en kontekst som kan ha stor betydning for undersøkelsens validitet. Kvale og Brinkmann (2015, s. 117) trekker frem at lydopptakeren kan bidra til å skape en «følelse av forlegenhet», og at intervjuet ofte går fra hyggelig småprat til en mer utspørrende karakter når lydopptakeren spiller inn samtalen. Dette kunne jeg til en viss grad også observere i intervjuene med de tre lærerne. At intervjuene var semistrukturerte kan imidlertid ha bidratt til at de fikk noe mer form av en samtale, der også intervjupersonene hadde en del av regien.

Jeg forsøkte bevisst å lytte til lærernes kommentarer og be dem utdype. Eksempler på at lærerne er både kritiske og positive, kan tyde på at situasjonen av trygg nok.

Robson og McCartan (2016, s. 170) peker på at validiteten er truet både i beskrivelsen, tolkningen og teorien til en fleksibel forskningsdesign. Her er det benyttet lydopptak og transkripsjon for å sikre en presis gjengivelse av intervjuene. Funnene er basert på min tolkning av resultatene og det teoretiske rammeverket som er benyttet i analysen, og det vil være en fare for at disse tolkningene og forklaringsmodellene har begrenset gyldighet. Jeg gjør i neste avsnitt rede for hvordan datamaterialet er analysert, for å sikre etterprøvbarhet, og jeg har forsøkt å knytte mine egne funn til tidligere forskning i den grad dette har vært tilgjengelig. Av hensyn til lærernes personvern, har jeg imidlertid valgt å bare publisere de relevante utdragene og ikke å inkludere de fullstendige transkripsjonene i sin helhet.

Analyse

Lydopptakene fra alle de tre intervjuene ble transkribert ord for ord på norsk bokmål med tradisjonelle norske tegnsettingsregler. Eventuelle kjennetegn ved dialekter er i all hovedsak fjernet. Lengre pauser eller at en person avbryter seg selv, er markert med ellipsetegn (tre prikker, …). Også småord som «ja», «mhm» og «eeh» ble inkludert i transkripsjonene, da disse for eksempel kan uttrykke at intervjupersonen er usikker. Sitatene som er gjengitt i den påfølgende resultatdelen har imidlertid blitt noe renskrevet for å øke lesbarheten og for å redusere at den skriftlige gjengivelsen av muntlig tale fremstår fordummende. Småord eller grammatiske feil er fjernet dersom de ikke anses å ha betydning for innholdet i utsagnet.

Etter at de tre samtalene var transkribert, ble lærernes utsagn kodet deduktivt etter de fire hovedkategoriene i rammeverket for algoritmisk tenkning til Weintrop et al. (2016), basert på kjennetegnene i Figur 2-3:

- Datahåndtering
- Modellering og simulering
- Digital problemløsning
- Systemtenkning

Kodingen gjorde det mulig å undersøke hva lærerne, individuelt og samlet, uttrykte om hver av de fire sidene ved algoritmisk tenkning. Uttalelsene innenfor hver av hovedkategoriene ble undersøkt for seg og sammenliknet med underkategoriene i rammeverket. På den måten var det mulig å se om det var noen mønstre i måten lærerne forstod algoritmisk tenkning som begrep.

Som Postholm (2005, ss. 99-100, 105) skriver, kan teori fungere som briller for forskeren i analysearbeidet og bidra til å få oversikt og se sammenhenger i materialet.

Videre ble det brukt tre koder for å markere lærernes holdninger til innføring av programmering i fysikkundervisningen:

- Læreren er positiv og ser fordeler
- Læreren er negativ og ser ulemper
- Læreren uttrykker usikkerhet

Ved å lese henholdsvis alle de positive, negative og usikre utsagnene samlet, var det enklere å samle de mulighetene og utfordringene lærerne så ved programmering i skolen, og det var mulig å undersøke hvorvidt lærerne var samstemte eller ikke.

Kodene er brukt som et verktøy i tolkningen og for å få oversikt over innholdet i intervjuet, men fremkommer ikke eksplisitt i resultatseksjonen.

3.2 Resultater

I dette delkapitlet presenterer jeg funnene som ble gjort i undersøkelsen. De tre intervjuene ble kodet individuelt, men resultatene presenteres sammenfattet og er oppsummert for å svare på forskningsspørsmålet. Delkapitlet er inndelt i overskrifter basert på funnene, ettersom lærerne er relativt samstemte.

Jeg vil i dette avsnittet vise at: (1) Lærerne definerer algoritmisk tenkning snevert, men anvender det vidt, (2) Lærerne opplever systemtenkning som et utfordrende begrep og (3) Lærerne er generelt positive til innføring av programmering i skolen, men de er usikre på om det vil gi økt læring og bidra til bedre fysikkundervisning.

Definerer algoritmisk tenkning snevert, anvender det vidt

Når lærerne blir bedt om å definere algoritmisk tenkning i starten av intervjuet, gir alle de tre forklaringer eller forsøk på definisjoner som minner om den snevrere definisjonen til Futschek (2006), altså det engelske begrepet «algorithmic thinking». Anders ser algoritmisk tenkning hovedsakelig som en problemløsningsstrategi og vektlegger overgangen fra et problem til et programmeringsspråk. Han mener imidlertid at det ikke er nødvendig å skrive faktisk programkode for å arbeide med algoritmisk tenkning – elevene kan skrive «pseudokode»:

> **Anders:** […] Tenke gjennom hvordan man kan finne oppskrifter til å løse problemet. […] At den gjør det regnestykket, og så få det over til programmeringsspråk som datamaskinen kan

> forstå. Og det kan godt være pseudokode, som du koder på arket. Og så… At de lærer å tenke algoritmisk, da.

Camilla har også en klar oppfatning av hva algoritmisk tenkning er, og hun vektlegger særlig rekkefølgen av operasjoner. Hun har også erfaring med at elevene i liten grad kan å tenke slik i dag, og hun ser det som positivt at elevene skal få mer opplæring i dette:

> **Camilla:** Hva det er, det vet jeg. Det er litt sånn å tenke at rekkefølge, eller ja, at ting skal gjøres i en presis rekkefølge for å nå riktig resultat. Og det jeg vet, er at elevene klarer ikke å tenke sånn. […]

Birgitte kjenner ikke til begrepet fra før og er mer usikker på hva det innebærer. Hun forstår også begrepet som en problemløsningsstrategi. Birgitte skiller seg imidlertid fra de andre lærerne, da hun også trekker inn at algoritmene kan illustreres ved at man henter inn resultater. Slik omtaler hun også kategorien «datahåndtering» i rammeverket, og ikke bare «digital problemløsning»:

> **Birgitte:** Ja, da tenker jeg matematisk prosedyre, tenker jeg, da. Egentlig. […] Jeg tenker jo det med… ehm… Altså det med at man har algoritmer, at man gjennomfører noe bestemte prosedyrer, at man regner på bestemte måter. […] Man kan kanskje bruke programmering til å illustrere … det … ved å ta inn en del resultat, da. Ved å bruke programmering til å illustrere de algoritmene, da, tenker jeg.

Når lærerne derimot omtaler hvordan de vil bruke programmering og algoritmisk tenkning i undervisningen, omtaler alle også datahåndtering, visualisering og modellering. På spørsmål om hvilke temaer han kan tenke seg å undervise med programmering i fysikk, svarer Anders at han vil bruke erfaring han har gjort i Teknologi og forskningslære:

> **Anders:** Det har jeg jo hatt i mange år. Der har vi jo programmert både med sånn blokkprogrammering i LEGO og Arduino-programvare. Hente data fra sensorer med telekommunikasjon, GPS og temperatur og trykk og andre sensorer.

Å knytte programmering til sensorer og datahåndtering tenker også de andre to lærerne. De mener at programmeringen fint kan knyttes til fysiske forsøk i klasserommet, og at observasjoner i forsøket kan sammenliknes med resultater fra digitale simuleringer. Alle

lærerne ser for seg å bruke programmering i mekanikk, og alle trekker frem beregning og visualisering av kastebanen i et skrått kast som et eksempel fra fysikklæreplanen. Birgitte nevner også elektromagnetiske bølger, og at det kanskje går an å gjøre noen beregninger med fotoner. Camilla omtaler hvordan man kan bruke visuell programmering og simuleringer til utforskning og leking for å lære flere begreper:

> **Camilla:** Ja, hvis de har lært å tegne ting, f.eks. med bølger og sånn, at de kan leke litt med hvordan avstanden mellom kildene påvirker interferensmønster, eller om det er mer i fase eller ikke i fase. At de kan leke mye mer selv enn det de kan gjøre praktisk nå, da. Og litt av det samme med alt som har å gjøre med mekanikken, for da kan vi veldig lett, hvis du endrer ett eller to tall, på en måte, på startinfoen du har, da kan du veldig fort se hva som skjer med banen. Det kan jeg bruke, gjerne, i undervisning. Det ser jeg for meg.
>
> **Intervjuer:** Ja, så mer visuelt enn numerisk.
>
> **Camilla:** Ja, mer visuelt. For jeg har erfart at de fleste elevene trenger faktisk å se ting, for å ha tro på at det er sånn. Og gjerne med at det er en film. At det må bevege seg. Det er ikke nok med et bilde. Da kan visuell programmering hjelpe til med det.

Systemtenkning er et utfordrende begrep

I den første delen av intervjuet, snakker lærerne i svært liten grad om praksisene som inngår i kategorien «systemtenkning». Ett unntak er når Anders forklarer hvordan han vil lage et opplegg med skrått kast. Da ser han for seg at elevene kan få utdelt noe ferdig kode og må utvikle en mindre del av programmet. Det omhandler å «utvikle digitale modulære løsninger» i hovedkategorien «digital problemløsning», men det innebærer også å forstå sammenhenger mellom ulike delsystemer, tenke i nivåer og håndtere kompleksitet. Dermed er Anders her inne på systemtenkning:

> **Anders:** Hvis man skulle ha en aktuell problemstilling hvor man skulle kaste ballen 10 meter og lande én meter over bakken, så kunne vi gitt dem den koden som skulle være rammen rundt, og lagd gulvet og den høyden over, og… Forskjellig. Tegne den kastemaskinen eller noe sånt i programmet. Mens de lager selve koden, da. At de må tenke: hva må vi putte inn for at den skal regne riktig kastebane, da. Ut fra de parameterne som er kjent.

Når lærerne eksplisitt spørres om hva systemtenkning kan innebære i algoritmisk tenkning i fysikk, blir derimot alle de tre usikre:

> **Anders:** Ja, det har det jo noe med hverandre å gjøre, selvfølgelig. Kan du… hva du tenker på med systemer. […] Jeg er litt usikker på hvordan… Hva du egentlig mener.

> **Birgitte:** Ja… Og hvordan programmering kan være med å gjøre til at man får… Elevene utvikler seg med tanke på systemtenkning. … Det har jeg ikke noe svar på.
>
> **Intervjuer:** Du har ikke noe tanker om, hva tenker du systemtenkning er?
>
> **Birgitte**: Nei, det er kanskje det som er litt vanskelig å… Jeg har vel ikke reflektert over hva som er systemtenkning. Det må jo handle om å kategorisere, da.
>
> **Intervjuer:** Sette ting i system, på en måte?
>
> **Birgitte:** Ja, rett og slett sette ting i system. […] Pluss at man bruker programmering til å få ting i system. Kanskje.

Camilla er den eneste som drar parallellen til systemer i fysikk, men hun sliter med å koble det til programmering og algoritmisk tenkning:

> **Camilla:** Det må… Ehh… Hva… System. Det forstår jeg ikke helt, hva det kan være i det… Det er… Når vi snakker om systemer når det gjelder bevegelsesmengde og sånn i fysikk, så har den tenkemåten med system å gjøre, men det klarer jeg nesten ikke å koble på digitalt, da. Eller programmering.

Positive til programmering, men usikre på læringen i fysikk

Som allerede nevnt, har alle lærerne erfaring med programmering, og de ser det som positivt at elevene skal lære programmering – både som fysikkelever og for allmenndannelsen. De er derimot mer skeptiske til at det bør innføres som en del av fysikkundervisningen, og de er usikre på om det egentlig kan gi bedre læring av fysikk. Dessuten frykter de at det vil ta for mye tid fra fysikklæringen. Anders tenker på fordeler med å lære algoritmisk tenkning som en problemløsningsstrategi når han blir spurt om det kan gi økt læring av fysikk:

> **Anders:** Ja, jeg tror det. Mhm. Men usikker på om det blir bedre eller ikke. Men jeg har veldig lyst til å prøve det ut. […] Jeg tror de kan bli flinkere til å strukturere, altså tankesettet eller hvordan de… Hva slags opplysninger de må putte inn i uttrykket sitt for å få det mest mulig

> effektivt. Og kanskje særlig i Fysikk 2, da. Som er mer regning. Du har et problem, og så har du en rekke med formler eller uttrykk som kan løse problemet.

Birgitte ser særlig fordelen med å forberede elevene til senere arbeidsliv og at programmering kan gi økt kompetanse i datahåndtering:

> **Birgitte:** Jeg tenker at det er kjempebra. Det er jo veldig… Altså, hvis man tenker på hvilke jobber de her skal ha, fysikkstudentene, så vil jo alle sammen trolig få jobber som involverer noen form for programmering. Det å lære seg å gjennomføre forsøk, og loggføre data, og kanskje få store mengder data, og bruke programmering til å undersøke dataene… Finne sammenhenger, det tror jeg er veldig… Veldig viktig. Og jeg tror det er kjempesmart at de begynner med det allerede nå.

Som vi allerede har sett, ser Camilla muligheter for programmering når det gjelder visualisering og simulering. Hun er derimot svært skeptisk til å innføre det som en del av fysikk-faget og frykter at det tar tid fra fysikklæringen uten å gi noe tilbake. I verste fall frykter hun at det blir dobbelt vanskelig:

> **Camilla:** Hvis de har erfaring med programmering fra før, og kan programmere fra før, da kan det være til hjelp, for da kan det åpne for flere muligheter i fysikken, da. Og kanskje litt mer på simulering og litt mer forståelse på fenomenene og sånn via simulering. Men hvis de ikke kan programmere fra før, som jeg mistenker det blir… Nei. Da ser jeg ikke noe som faktisk er til hjelp. Jeg ser for meg mer at det blir noe i tillegg som vil bli en hindring. For du må faktisk ganske langt, og kunne ganske mye programmering, for å kunne bruke det som hjelp til å forstå noe annet. Det… det har jeg følelse av.

Hun har problemer med å finne problemstillinger som er enkle nok til at fysikkelevene kan løse dem med programmering. Hun frykter at innføring av programmering kan gjøre faget mer kvantitativt og ta vekk noe av det kvalitative preget fagene har i dag:

> **Camilla:** Og jeg krysser fingrene for at vi ikke ber dem om at vi får det til, for da… da får vi trøbbel! […] Og da vet jeg at alle elevene vil spørre: Men hvorfor gjør vi det så tungvint når vi bare kan trykke på en knapp i Excel eller GeoGebra?!

3.3 Oppsummering og diskusjon

Denne undersøkelsen er nokså liten, og tre lærere fra samme skole er ikke nok til å gi et generelt bilde av fysikklæreres syn på programmering og algoritmisk tenkning. I dette avsnittet vil jeg likevel forsøke å trekke noen implikasjoner fra resultatene. Det er naturlig å tro at de oppfatningene lærerne uttrykker her, også deles av andre lærere. Dermed bør vi likevel kunne si noe generelt om hvordan fysikklærere forstår begrepet «algoritmisk tenkning» og hvilke muligheter og utfordringer som ligger i innføringen av programmering i fysikkundervisningen.

Resultatene viser at lærerne bruker nokså smale definisjoner når de blir bedt om å definere algoritmisk tenkning. Det kan være et problem at Utdanningsdirektoratet innfører dette begrepet i læreplanene, dersom lærerne feiltolker begrepet til algoritmeutvikling fremfor den bredere intensjonen som ligger bak begrepet «computational thinking» (Utdanningsdirektoratet, 2019a; Wing, 2006). I verste fall mister begrepet hele sitt allmenndannende preg, og vi kan risikere at elevene ender med å arbeide med svært snever algoritmeutvikling som bør være spesielt forbeholdt informatikere. Skolemyndighetene og læreplanforfatterne bør derfor være spesielt tydelige på hvordan de definerer «algoritmisk tenkning» for å unngå misforståelser. Eventuelt bør det også vurderes å finne en bedre norsk oversettelse. Jeg har i kapittel 2.2 foreslått begrepet «datalogisk tenkning» som et mulig alternativ.

Selv om lærerne altså forstår selve begrepet «algoritmisk tenkning» ganske snevert, viser resultatene at de likevel trekker inn flere sider ved taksonomien til Weintrop et al. (2016) når de snakker om hvordan programmering kan implementeres i praktisk undervisning. De kommer alle med eksempler på undervisningsopplegg som inneholder datahåndtering, visualisering, simulering og problemløsning. Blant temaene som undervises i dagens læreplan, er det spesielt hovedområdet mekanikk som trekkes frem. Det er også innenfor disse områdene at lærerne mener programmering og algoritmisk tenkemåte eventuelt kan bidra til økt læring av selve fysikken. Birgitte peker også på at det kan være nyttig å lære fysikkelevene programmering som verktøy, med tanke på fremtidig arbeidsliv.

Samtidig kan vi, som Camilla også gjør, spørre oss om det er nødvendig å lære programmering for å håndtere og visualisere data. Elevene gjør dette allerede i dag med programmer som Excel og GeoGebra, og de bruker ferdige animasjoner og simuleringer i innlæring av fysikkbegreper, uten å kunne kode. Dersom elevene implementerer koden selv, vil de nødvendigvis komme nærmere datamaterialet, men om det er verdt tiden det tar å lære så

omfattende programmering som kreves, og om det vil gi økt læring av fysikkinnholdet, er mer usikkert. Det er egentlig påfallende hvor få tilfeller lærerne ser for seg at programmering kan være nyttig, til tross for at de alle har erfaring med programmering i egen utdanning og annet arbeid. En kunne tenke seg at de for eksempel så muligheter i numerisk likningsløsning, derivasjon og integrasjon. Som vi har sett hos blant annet Malthe-Sørenssen et al. (2015) og diSessa (2018), kan dette gjøre det mulig å løse mer virkelighetsnære problemer der matematikken ellers ville vært for vanskelig. Den eneste som eksplisitt nevner numeriske beregninger blant de intervjuede lærerne, er Camilla, men hun tenker nokså avansert og «krysser fingrene» for at dette ikke skal innføres.

Vi ser også at lærerne ikke klarer å koble fysikkinnhold og programmering til systemtenkning – det høyeste nivået i taksonomien til Weintrop et al. (2016). Dette kan skyldes at denne delen av taksonomien ikke passer så godt inn i fysikkfaget, men det kan nødvendigvis også ha sammenheng med at lærerne i liten grad har tenkt på hvordan programmering kan innføres i fysikk før intervjuet. De er enda usikre på hvordan læreplanene vil se ut. Til sammen viser dette likevel at læreplanene i fysikk bør gjøre ganske tydelige valg av innhold som egner seg til programmering og algoritmisk tenkning. Læreplanene kan med fordel peke på hvilke temaer og arbeidsmåter som spesielt er egnet, og gi veiledning om hvordan programmering i praksis kan integreres i fysikkfaget.

Lærerne uttrykker seg positivt til programmering og ser på det som en viktig kompetanse elevene bør ha. De er nysgjerrige og har lyst til å prøve det ut i sin egen undervisning. De har også tro på at det er mulig å finne områder der algoritmisk tenkning kan bidra til økt læring i fysikk, selv om de har problemer med å finne særlig mange eksempler selv.

Camilla er tydeligst på at det kan bli dobbelt så vanskelig å lære programmering sammen med matematikken og fysikken, i stedet for å innføre et eget programmeringsfag. Lærerne frykter også at det vil bli for liten tid og at elevene vil møte med svært ulike programmeringsferdigheter. Vi ser altså at lærerne peker på mange av de samme utfordringene som også kommer frem i litteraturstudien (For eksempel Caballero et al., 2013; Sørby og Angell, 2012; Taub et al., 2015). I Sverige har innføringen av programmering i skolen kommet noe lenger enn i Norge. I en tilsvarende studie, gjort av Nilsson (2018) blant svenske matematikklærere, finner han flere av de samme bekymringene som fysikklærerne i denne undersøkelsen uttrykker. Hvis elevene ikke har opparbeidet seg tilstrekkelige programmeringsferdigheter fra tidligere skoleår, vil fysikklærerne risikere å måtte starte forfra. Det spørs da om elevene vil komme opp på et høyt nok nivå til at programmeringen faktisk kan

ha noe for seg i fysikken. Det kreves en viss koordinasjon mellom fysikk- og matematikk-fagene, og mellom videregående skole og grunnskole. En mulig løsning kan være å finne opplegg som har en enkel inngang, men som gir mulighet for større utfordringer og å utforske problemet grundigere i dybden. Sengupta et al. (2013, s. 362) uttrykker det ved at læringsaktiviteter med algoritmisk tenkning bør ha «en lav terskel og et høyt tak». Her blir det viktig at lærerne får tid og veiledning nok til å tenke ut og utforme gode undervisningsopplegg som passer elevenes nivå, og som har en tilstrekkelig kobling til fysikkinnhold.

For videre forskning kan det være interessant å gjennomføre den samme undersøkelsen blant flere lærere, på flere skoler og i flere fag, og kanskje også utvide med en spørreundersøkelse, for å undersøke om forståelsen av den algoritmiske tenkningen er den samme også i andre lærergrupper. Etter hvert som de nye læreplanene skrives og lærerne blir bedre kjent med innholdet, vil det også være interessant om man kan observere en endret forståelse av begrepet og mulighetene programmering kan ha i undervisningen.

4 UNDERVISNINGSOPPLEGG: Utvikling og utprøving

Med utgangspunkt i litteraturstudien i kapittel 2 og intervjuene med fysikklærerne i kapittel 3, har jeg utviklet tre undervisningsopplegg med programmering og algoritmisk tenkning for fysikk på videregående skole. Dette kapitlet utgjør dermed utviklingskomponenten av det designbaserte forskningsarbeidet. De tre oppleggene er testet ut på elever med observasjon og spørreundersøkelse for å besvare følgende forskningsspørsmål:

3. Hvordan kan et undervisningsopplegg med programmering se ut?
    a. Hvor utfordrende opplever elevene opplegget?
    b. Hva opplever elevene som utfordrende?
    c. Ser elevene relevansen til fysikk?

Jeg presenterer intensjonene og bakgrunnen for undervisningsoppleggene i kapittel 4.1, og begrunnelser for valg av programmeringsspråk gis i kapittel 4.2. Undervisningsoppleggene presenteres og begrunnes i kapittel 4.3. I kapittel 4.4 presenterer jeg hvordan oppleggene er prøvd ut i klasserommet, metode for datainnsamling og resultatene fra utprøvingen. Undervisningsmaterialet som ble brukt i klasserommet er lagt ved som vedlegg. Oppgaveark som ble utdelt til elevene, finnes i vedlegg D, E og F. Lysbildepresentasjoner som ble vist for elevene, ligger i vedlegg G, H og I. Datafiler, programkode og ferdige løsningsforslag finnes i vedlegg J.

4.1 Intensjon og prinsipper

Programmering finnes ikke i den nåværende læreplanen for fysikk, og vi vet enda ikke hva som kommer i den nye læreplanen. Skissene til nye læreplaner i matematikk (Utdanningsdirektoratet, 2018b, 2019c), har imidlertid gitt en indikasjon på hvilke programmeringsferdigheter som kan være aktuelle. Formålet med undervisningsoppleggene har vært å forsøke å integrere programmering og algoritmisk tenkning med kompetansemålene som finnes i dagens læreplan for fysikk på videregående skole. Det gir noen begrensninger:

Elevene kan ikke programmering fra før, og trenger en innføring i dette, og det er begrenset hvor mye tid vi kan bruke til utprøving, ettersom elevene har andre kompetansemål de må arbeide med. Samtidig gir det mulighet til å undersøke hvor raskt programmering med fysikkinnhold kan introduseres, og i hvor stor grad dagens læreplan har innhold som egner seg for programmering. Vi har sett at mange av eksemplene fra litteraturstudien er hentet fra mekanikk. Det har derfor også vært et mål å finne opplegg som knytter seg til andre områder i læreplanen.

Når den nye læreplanen har virket en stund, vil fysikkelevene antakelig kunne en del programmering fra før, gjennom opplæring i grunnskolen og gjennom matematikk-faget på videregående skole. Formålet med undervisningen her er ikke først og fremst at elevene skal bli drevne programmerere i løpet av utprøvingen, men at programmeringen skal kunne bidra til læring av fysikk og en dypere forståelse for begreper og fenomener de kjenner fra før. Utfordringen blir dermed å kombinere innføring i enkel programmering med noe mer avansert fysikk. Samtidig er det et mål at elevene får erfaring med hvordan fysikk anvendes i arbeid og ingeniørvitenskap, og at de skal se at programmering kan være nyttig både for fysikkfaget og for samfunnet.

I utformingen av undervisningsoppleggene har jeg tatt utgangspunkt i praksisene for algoritmisk tenkning i taksonomien til Weintrop et al. (2016), som er gjengitt og omtalt i kapittel 2.2. Det er ikke mulig å implementere alle praksisene i like stor grad i løpet av et kort undervisningsopplegg, så det er forsøkt å legge vekt på ulike deler av taksonomien i de ulike oppleggene.

Det er forsøkt å inkludere flere av prinsippene for undervisning med algoritmisk tenkning som er omtalt i kapittel 2.5. Det er en fordel om oppgavene kan tilpasse seg selv. Oppgavene elevene får bør, med terminologien til Sengupta et al. (2013), ha en så lav terskel at alle kommer i gang, helst uten særlig støtte fra læreren. Når det er mange elever i en klasse, vil ikke læreren ha tid til å hjelpe alle i gang. Det er derfor en fordel om de første oppgavene i større grad «holder elevene i hånda» og kanskje likner et tidligere gitt eksempel, slik blant annet Sørby (2010) kommenterer. Elevene bør få erfaring med å skrive kode selv, og dermed arbeide med sin egen datamaskin. Likevel bør oppgavene egne seg til samarbeid slik at elevene kan hjelpe hverandre og diskutere og reflektere sammen, slik Hutchins et al. (2018) oppdaget at elevene gjerne gjør. Med fagfornyelsen er «dybdelæring» et viktig prinsipp. Derfor bør undervisningsoppleggene også ha noen problemløsningsoppgaver som er egnet for å utforske videre og gir elevene muligheter til å tenke algoritmisk. Det er forsøkt å lage oppgaver som er knyttet til fysiske situasjoner som har rot i virkeligheten, eller oppgaver som visualiserer og

simulerer fenomener som ikke lar seg observere på fysikklaben. Det kan bidra til at elevene ser programmering som relevant for kommende studier og arbeidsliv, og som Brown og Wilson (2018) påpeker, kan autentiske oppgaver bidra til å vekke interesse. Læreren får en sentral rolle som modell og støtte under programmeringen, og som leder av klassediskusjoner, i samsvar med rådene fra blant annet Brown og Wilson (2018).

Intervjuene med lærerne i kapittel 3 har også vært et viktig grunnlag for utviklingen av oppleggene. Lærerne hadde flere ideer til hvordan programmering kunne innføres i ulike fysikktema. Deres tanker om å knytte programmering til datastrømmer fra sensorer, konkrete eksempler innenfor mekanikk, fotoner og bølgefysikk, har vært til inspirasjon. Hvordan lærerne understreker betydningen av visualisering og simulering, har også hatt betydning for flere av oppleggene. Samtidig har jeg også forsøkt å gi oppleggene innfallsvinkler som lærerne selv ikke har tenkt på, og bruke flere elementer fra taksonomien til Weintrop et al. (2016), slik at oppleggene kan bidra til et noe utvidet perspektiv på programmering på fysikkfagets premisser. Jeg har blant annet forsøkt å konkretisere hvordan «systemtenkning» kan inngå i undervisningen gjennom det Orban et al. (2018) kaller en hybrid tilnærming. Jeg har også forsøkt å inkludere numeriske beregninger på en enklest mulig måte, med blant annet Eulers metode og «ny = gammel + endring» (diSessa, 2018; Yaşar, 2013), slik at det har en praktisk anvendelse uten å gi kognitiv overbelastning (Taub et al., 2015). Oppleggene har også hentet inspirasjon fra eksemplene som forekommer i litteraturen, blant annet eksemplene om Usain Bolt hos Malthe-Sørenssen et al. (2015), ladninger i elektromagnetisk felt hos Kortemeyer og Kortemeyer (2018) og andregradsformelen hos Tellefsen (2018).

4.2 Valg av programmeringsspråk

Det finnes en rekke ulike programmeringsspråk, og skolene må gjøre et valg om hvilket språk elevene skal lære seg. Slik utkastene til læreplanen i matematikk på videregående nivå ser ut (Utdanningsdirektoratet, 2018b, 2019c), er blokkprogrammering ikke særlig egnet. Når programmene blir større og det kreves en del linjer, er det mest ryddig å bruke et tekstbasert programmeringsspråk.

Jeg har i dette arbeidet valgt å bruke Python. Selv om det finnes andre språk som antakelig gjør jobben like godt, har Python noen klare fordeler. Først og fremst er det en relativt enkel syntaks, som gjør at det er relativt raskt å lære, enkelt å tolke og intuitivt å forstå. Samtidig har Python en del avanserte muligheter, særlig ved at man kan inkludere en del tilleggspakker. Det gjør det egnet for også å lage visuelle fremstillinger, grafer og animasjoner, og å utføre mer

avanserte matematiske beregninger. Python er dessuten gratis, enkelt å installere, og er også mye brukt i faktisk (vitenskapelig) programmering. Fysikkstudenter ved både UiO og NTNU får opplæring i Python, og det er relativt enkelt å overføre prinsippene til andre programmeringsspråk senere. Elever som lærer Python, får altså opplæring i et programmeringsspråk som faktisk brukes av fysikere, men har også muligheter til å lage egne spill og andre programmer med det samme språket. Yaşar (2013, s. 15) valgte med mange av de samme grunnene å bytte fra andre programmeringsspråk til Python.

Elevene ble i dette forsøket bedt om å installere «Anaconda», en pakke som inneholder Python og alle de nødvendige tilleggspakkene og programmene. Innstalleringen er rett frem, men pakken er nokså stor, så det kan ta en del tid. Med Anaconda følger programmet «Spyder», som er editoren som brukes for å skrive og kjøre Python-kode.

I Python-programmene benytter jeg pakken «pylab», som følger med Anaconda-installasjonen. Denne pakken inneholder blant annet en del matematiske verktøy og muliggjør tegning av grafer. Tilsvarende verktøy finnes i pakkene «numpy» og «matplotlib», men «pylab» forenkler anvendelsen. For animasjon brukes her pakken «tkinter». Elevene lager ikke selve animasjonene selv i disse oppleggene, men bruker ferdige elementer.

4.3 Undervisningsopplegg

I denne seksjonen presenteres de tre undervisningsoppleggene. For leserens oversikt inneholder hvert opplegg en kortfattet introduksjon om innholdet, beregnet tid for gjennomføring og nødvendig utstyr. Det henvises til noen relevante kompetansemål fra dagens læreplaner i fysikk som berøres av opplegget, og det er laget noen mer konkrete læringsmål for undervisningen. Hvert opplegg har en punktvis oversikt over timens forløp og struktur, som etterfølges av mer detaljerte beskrivelser og begrunnelser. I oppleggene henvises det til aktuelle lysbilder fra vedleggene. Lysbildepresentasjonen vises for elevene i timen, foruten når læreren skriver kode i Spyder.

Oppleggene presenteres i sin opprinnelige form og er ikke justert etter utprøving. Unntaket er tidsestimatene, som har blitt noe forlenget, slik at de bedre gjenspeiler den faktiske tidsbruken. Oppleggene presenteres altså her som et første steg i det som kunne vært en lengre iterativ prosess med ytterligere justeringer og forbedringer, blant annet basert på erfaringene som presenteres i kapittel 4.4.

OPPLEGG 1: Analysere rettlinjet bevegelse (Fysikk 1)

Dette undervisningsopplegget omhandler rettlinjet bevegelse, sammenhengen mellom strekning, fart og akselerasjon, og numerisk beregning av den deriverte. Elevene får trening i å håndtere data og bruker programmering til å analysere og visualisere disse. Elevene får utdelt ferdige datafiler og gjør også et eget forsøk for å generere egne data. Dette opplegget passer for Fysikk 1.

TID: 5 timer (à 45 minutter)

UTSTYR:

- Datamaskiner med installerte programvarer Python/Anaconda og Tracker
- Tekstfil med posisjonsdata for Warholms 400-meter hekk (`Warholm400.txt`)
- Tekstfil med fartsdata for en lærers løpetur (`Løpetur.txt`)
- Utstyr for å lage rettlinjet bevegelse i fysikk-laben. F.eks. fjær, kloss, skråplan, bil, …
- Kamera (smarttelefon)

KOMPETANSEMÅL (FRA LÆREPLANEN I FYSIKK 1)**:**

- bruke parameterframstilling til å beskrive rettlinjet bevegelse for en partikkel, og bruke derivasjon til å regne ut fart og akselerasjon når posisjonen er kjent, både med og uten digitale verktøy
- lage en eller flere matematiske modeller for sammenhenger mellom fysiske størrelser som er funnet eksperimentelt
- samle inn og bearbeide data og presentere og vurdere resultater og konklusjoner av forsøk og undersøkelser, med og uten digitale verktøy

LÆRINGSMÅL:

- Gjøre numeriske beregninger av posisjon, fart og akselerasjon i rettlinjet bevegelse
- Samle, håndtere, analysere og visualisere data i filer med programmering
- Diskutere numeriske feil
- Anvende numerisk derivasjon i problemløsning

STRUKTUR:

- *Oppstart*: Presentasjon av mål, programmering og algoritmisk tenkning
- *Gjennomgang*: Løkker og numerisk derivasjon
- *Felles eksempel*: Farten i Warholms hekkeløp
- *Oppgave*: Akselerasjonen i Warholms hekkeløp
- *Klassediskusjon*: Om gode datamateriale og tolkning av resultatene
- *Gjennomgang*: Analysere og eksportere data med Tracker
- *Oppgave*: Filme og analysere en rettlinjet bevegelse
- *Oppgave*: Finne lengden av lærerens joggetur

Etter at elevene har lastet ned og installert den nødvendige programvaren, starter timen med en lærerpresentasjon om hvorfor elevene skal lære programmering og algoritmisk tenkning, slik at elevene forstår hvorfor de skal bruke tid på å lære dette, og forhåpentligvis motiveres ved å se at det er nyttig for fysikere spesielt og for samfunnet generelt (Lysbilde G-3). Læreren presenterer også læringsmålene for timen, og innfører gjennom disse begrepene «programmeringsspråk» og «algoritmisk tenkning» (Lysbilde G-4).

I Lysbilde G-5 presenteres metoden for numerisk derivasjon fra kapittel 2.6, å bruke gjennomsnittlig vekstfart mellom to nærliggende punkter, og den sammenliknes med den analytiske definisjonen av derivasjon som elevene kjenner fra tidligere. Det vektlegges at det er en tilnærming som blir stadig bedre hvis punktene ligger nærmere hverandre. Matematikken knyttes til fysikk ved å bruke symboler $v(t)$ og $s(t)$ for fart og strekning i stedet for en mer generell $f(x)$. I Lysbilde G-6 forlater læreren presentasjonen og går gjennom et eksempel med elevene. Læreren demonstrerer hvordan et nytt Python-program opprettes i Spyder og skriver den første programkoden sammen med elevene. I datafilen `Warholm400.txt` finnes data over hvor lang tid hekkeløperen Karsten Warholm hadde brukt ved hvert hinder i et 400 meter langt hekkeløp. Læreren demonstrerer hvordan elevene kan laste inn filen i programkoden og hente ut dataene fra de ulike kolonnene. Deretter vises det hvordan dataene kan illustreres i en graf, og hvordan en kan regne ut fart numerisk fra posisjonsdataene. Elevene får så i oppgave å utvide programkoden på egen hånd, slik at den regner ut akselerasjonen til Warholm og tegner akselerasjonsgrafen. Elevene skriver koden på sin egen maskin, men samarbeider gjerne i mindre grupper. De følger samme strategi som ved utregningen av fart, men må gjøre noen mindre endringer i koden. På denne måten trenger ikke elevene å huske eller forstå syntaksen

helt, men blir samtidig kjent med kodespråket og de ulike elementene i koden. Den ferdige akselerasjonsgrafen blir nokså «hakkete» på grunn av få målepunkter og usikkerhet i datamaterialet. Det kan se ut til at Warholm akselererer kraftig opp og ned, men ved å studere akseverdiene kan elevene oppdage at fluktuasjonene er små. Den første oppgaven bør ende i en klassediskusjon om hva som er gode data for numerisk derivasjon og underbygge lærerens introduksjon om at det bør være mange nok punkter som ligger tett.

I denne første delen av undervisningsopplegget blir elevene gjort kjent med programmeringsspråket og Spyder, de lager et enkelt program og får antakelig allerede erfare feil i koden som må finnes og rettes. Dessuten vil de både behandle, analysere og visualisere data lagret i en ekstern fil. Dermed arbeider de allerede med flere av praksisene innenfor «datahåndtering» og «digital problemløsning» i taksonomien over algoritmisk tenkning (kapittel 2.2, s. 28). Dette gjøres med mye veiledning og støtte. Det er helt nødvendig at elevene får et eksempel og en demonstrasjon for å komme i gang. Selve matematikken og fysikken bør være kjent for elevene, men måten det kombineres på, i et helt nytt programmeringsspråk og med mange nye tekniske operasjoner, kan gjøre at det likevel oppleves vanskelig for elevene.

I den neste delen av undervisningsopplegget, presenteres elevene for programmet Tracker (Lysbilde G-9). Dette er et program som kan brukes til å analysere en videofilmet bevegelse. Læreren demonstrerer med en eksempel-video, for eksempel en fallende gjenstand. Elevene må vises hvordan Tracker kan kalibreres ved å bruke et objekt med kjent lengde i samme plan som den fallende gjenstanden, hvordan bevegelsen kan spores som en punktmasse, og hvordan posisjonsdataene kan eksporteres fra Tracker til en tekstfil. Det er viktig at tallformatet i Tracker settes til å bruke punktum som desimaltegn før dataene eksporteres, slik at Python tolker tallene riktig.

Deretter er det elevenes oppgave å bruke utstyr de finner på fysikk-laben, som for eksempel fallende gjenstander, lekebiler, elever som hopper eller løper, masser i fjær, eller annet for å filme og analysere en rettlinjet bevegelse. Videoen legger de inn i Tracker-programmet, som de bruker til å lage en tekstfil med tids- og posisjonsdata. De laster dataene inn i Python og henter ut de relevante datakolonnene for å analysere bevegelsen og tegne grafer. Oppgaven ber også elevene om å reflektere over hva dataene de har funnet kan si om krefter og energi i bevegelsen.

I denne delen av undervisningsopplegget arbeider elevene fortsatt med de samme prinsippene for numerisk derivasjon i Python. Dermed bør de nå bli mer fortrolige med koden, og de bør klare å tilpasse den til de nye datafilene. Nå utvides imidlertid oppgaven til også å

inneholde innsamling og generering av egne data. Det at elevene selv har utført og filmet bevegelsen, kan gjøre det lettere å knytte de resulterende grafene til bevegelsen og tolke innholdet i grafene. Dessuten trekkes det inn flere fysikkelementer, ved at elevene blir bedt om å bruke datamaterialet til å finne krefter og energi. Denne delen dreier seg fortsatt i stor grad om «datahåndtering», men innebærer nå noe mer «digital problemløsning» ettersom elevene nå i større grad må håndtere programmeringen, forberede den fysiske bevegelsen slik at den kan analyseres digitalt, velge beregningsverktøy og overføre funnene til andre fysiske begreper. Lærerens rolle blir nå i enda større grad å veilede og å stimulere elevene til å tenke algoritmisk og finne en strategi for å løse utfordringen. Samtidig kan elevene også hjelpe hverandre.

I den siste delen av undervisningsopplegget (Lysbilde G-12) får elevene utdelt en ny datafil (`Løpetur.txt`) med informasjon om en lærers joggetur. Den skiller seg fra de andre datafilene ved at den ikke oppgir tid og posisjon, men derimot tid og fart. I tillegg er farten oppgitt som antall hele minutter og sekunder per kilometer, i stedet for den mer vanlige meter per sekund. Denne oppgaven er dermed mer utfordrende, ved at den først krever at elevene forstår hvilke data som er gitt, og deretter henter dem ut på en måte slik at de kan brukes til å løse den siste utfordringen: Å finne ut hvor lang løpeturen var. Her får elevene mindre veiledning enn tidligere, og de må selv finne en strategi for å løse problemet. For å regne ut strekningen, må de kunne snu på formelen for numerisk derivasjon og implementere denne på riktig måte i programkoden.

Denne delen av opplegget har større fokus på problemløsning. Elevene skal ikke lenger bare følge den samme oppskriften som tidligere, men må faktisk forstå hvilke data de arbeider med. Dette vil være ekstra utfordrende, særlig for elever som ikke er vant til å «tenke algoritmisk». Læreren får en ekstra viktig rolle som støtte, og det kan være nødvendig å lede elevene på rett spor. Samtidig er det viktig at læreren ikke gir hele svaret for raskt, da elevene nettopp skal trenes i problemløsning. I første omgang bør læreren hjelpe elevene med å tolke hvilke data de er gitt, for så å lede dem i retning av hvordan disse kan brukes til å beregne farten i meter per sekund. Elevene trenger antakelig også hjelp med å implementere formelen for strekning i programkoden, $s_{i+1} = s_i + v_i \cdot t_i$. Denne er strengt tatt en form for numerisk integrasjon, men kan her betraktes som en omskriving av formelen for numerisk derivasjon, eller en generalisering av bevegelsesformelen ved konstant fart, $s = s_0 + v \cdot t$.

OPPLEGG 2: Numerisk beregning av bevegelse i inhomogent felt (Fysikk 2)

Dette undervisningsopplegget omhandler en ladet partikkels bevegelse i det inhomogene elektriske feltet rundt én eller flere andre ladede partikler. Elevene regner med krefter, Newtons andre lov og numerisk integrasjon, og de programmerer funksjoner og mindre komponenter som til slutt settes sammen for å animere bevegelsen. Opplegget omhandler elektrisitet, men selve håndteringen av krefter og bevegelse er likevel mekanikk. Opplegget passer for Fysikk 2.

TID: 5 timer (à 45 minutter)

UTSTYR:

- Datamaskiner med installert programvare Python/Anaconda
- Delvis ferdig kode som styrer animasjonen (`Ladning.py`)

KOMPETANSEMÅL (FRA LÆREPLANEN I FYSIKK 2)**:**

- beskrive homogene og inhomogene elektriske felt og bruke Coulombs lov
- beskrive banen til en partikkel ved hjelp av parameterframstilling, og bruke derivasjon og integralregning til å regne ut posisjon, fart og akselerasjon når en av de tre størrelsene er kjent

LÆRINGSMÅL:

- Bruke Coulombs lov til å beregne krefter mellom partikler
- Lage funksjoner og sette dem sammen til et helhetlig system
- Bruke Newtons andre lov og Eulers metode for integrering til å beskrive en partikkels bevegelse i inhomogene elektriske felt

STRUKTUR:

- *Oppstart*: Presentasjon av mål, programmering og algoritmisk tenkning
- *Gjennomgang*: Funksjonsbegrepet i programmering
- *Felles eksempel*: Lage en akselerasjonsfunksjon som avhenger av tid
- *Gjennomgang*: Løkker og numerisk integrasjon
- *Felles eksempel:* Beregne fart og strekning fra akselerasjonsfunksjonen numerisk
- *Oppgave:* Lage funksjon for å beregne avstand
- *Oppgave:* Lage funksjon for å beregne elektrisk kraft
- *Oppgave:* Bruke funksjonene og numerisk integrasjon i simulering
- *Oppgave:* Utforske bevegelse i inhomogent elektrisk felt
- *Klassediskusjon:* Om bevegelse i inhomogene felt og numeriske feil

Timen starter med en presentasjon av programmeringens rolle i fysikk og i samfunnet forøvrig for å motivere for undervisningen (Lysbilde H-3). Læreren presenterer læringsmålene for timen, og gjennom disse presenteres begrepene «programmeringsspråk» og «algoritmisk tenkning» (Lysbilde H-4).

I Lysbilde H-5 introduseres elevene til funksjonsbegrepet i Python. Det er gjort ved å knytte det til funksjoner i matematikk, som allerede er kjent for elevene. I tillegg til å gjøre funksjoner i Python litt mindre fremmed, kan det kanskje også bidra til at elevene får en utvidet forståelse av funksjonsbegrepet i matematikk (som noe mer enn en likning eller en graf). Læreren åpner deretter Spyder og programmerer en enkel funksjon sammen med elevene. Funksjonen tar inn en tid t og returnerer en verdi for akselerasjon som inneholder t^2. Dermed lærer elevene også den litt uvanlige syntaksen for potenser i Python, som skrives `t**2`, og ikke med symbolet `^` som er vanlig i andre språk og kalkulatorer.

Til slutt i lærerens gjennomgang (Lysbilde H-6) presenteres Euler-metoden for numerisk integrasjon som «ny = gammel + endring», som omtalt i kapittel 2.6. Det vises hvordan strekning kan beregnes fra fart, og hvordan fart kan beregnes fra akselerasjon, ved å sammenlikne med bevegelsesformlene for henholdsvis konstant fart og akselerasjon. Læreren vektlegger at det må være svært små tidssteg mellom hver beregning. Grafen i lysbildet illustrerer at beregningen fungerer nokså bra med litt store tidssteg, og læreren understreker at med enda mindre steg, ville den beregnede grafen blitt enda riktigere. Læreren utvider deretter eksempelkoden sammen med elevene, slik at akselerasjonsfunksjonen brukes til å beregne fart

og posisjon i en `while`-løkke. Gjennom dette eksempelet introduseres også variabel-begrepet og `print`-kommandoen for utskrift til skjerm. Læreren må gjerne legge inn en `print`-kommando inne i `while`-løkka, for å vise hvordan datamaskinen regner ut svært mange posisjoner på svært kort tid.

I resten av opplegget arbeider elevene med oppgaver som gradvis bygger opp en ferdig simulering. Opplegget følger dermed det Orban et al. (2018) kaller en «hybrid tilnærming», som omtalt i kapittel 2.5. Elevene utvikler de komponentene av simuleringen som er sentrale for fysikkfaget, mens de mer avanserte og tekniske elementene er ferdiglaget.

I den første oppgaven (Lysbilde H-8) lager elevene en funksjon for å beregne avstanden mellom to punkter i koordinatsystemet. Det er viktig at elevene her får prøve å løse problemet selv og oppfordres til å tegne, tenke logisk og samarbeide, mens læreren fungerer som en støtte i diskusjonen. Elevene trenger egentlig bare å bruke pythagorassetningen for å løse problemet, men å bruke denne i det nye programmeringsmiljøet kan virke utfordrende. Hvis elevene likevel oppfordres til å tegne og løse problemet uten programmering først, vil det kanskje være enklere å implementere det i kode etterpå. Elevene bør teste at funksjonen fungerer som den skal. Det er en viktig feilsøkingsstrategi å teste mindre komponenter før de settes sammen til en større helhet.

I den andre oppgaven (Lysbilde H-10) lager elevene en ny funksjon som regner ut den elektriske kraften på en ladning A fra en annen ladning B. Funksjonen skal returnere kraftkomponentene som en vektor. Det tekniske i syntaksen er oppgitt for elevene, slik at de kan fokusere på selve fysikkinnholdet. Her må elevene lete frem formelen for Coulombs lov og implementere den i koden. Dessuten må de løse problemet å dele opp kraften i komponenter. Igjen bør elevene oppfordres til å tegne, tenke og samarbeide, og det kan være naturlig for læreren å gi noen hint om definisjonene av cosinus og sinus med kateter og hypotenuser.

Når funksjonene for avstand og coulomb-kraft er ferdige, kan elevene laste ned filen `Ladning.py` (Lysbilde H-12). Denne filen inneholder en del teknisk kode for å vise animasjon i Python, som er for vanskelig til at elevene kan kode selv på nåværende stadium. Elevene bør likevel kunne åpne koden og gjøre ferdig mindre deler av den. Blant annet kan elevene legge inn de to funksjonene for avstand og kraft, og de kan legge inn tallverdier for konstanter, masser og startposisjoner (Lysbilde H-13). Dermed arbeider elevene bare med mindre komponenter av et større system, og trenger ikke forstå alle detaljer ved hele systemet.

Samtidig forstår de virkningen av systemet som helhet. De arbeider altså med flere av praksisene innenfor «systemtenkning» i taksonomien over algoritmisk tenkning (Figur 2-3).

I Lysbilde H-15 blir elevene bedt om å bruke funksjonen for coulomb-kraften til å beregne akselerasjon, fart og posisjon. Elevene må her anvende Newtons andre lov og teknikkene for numerisk integrasjon. Dette brukes til å styre animasjonen, og når dette er implementert, kan elevene kjøre koden (Lysbilde H-17) og forhåpentligvis se at partikkelen beveger seg. I den resterende tiden blir elevene bedt om å utforske fysikken som nå blir synlig gjennom simuleringen. De kan endre på ulike startparametere, forsøke å få partikkelen til å gå i sirkelbane, legge inn flere partikler med ulike ladninger og få partikkelen til å oscillere frem og tilbake. Simuleringen visualiserer objekter som ellers ville vært vanskelig å studere på laben, og elevene kan kvalitativt studere bevegelsen i det inhomogene feltet. Her berøres flere av praksisene innenfor «modellering og simulering» i taksonomien over algoritmisk tenkning (Figur 2-3). Det er imidlertid viktig å være klar over at beregningene vil være noe feilaktige hvis partiklene kommer nokså nære hverandre, for da blir kreftene store i forhold til tidsstegene. Det kan åpne for diskusjoner om valg av steglengder og begrensninger ved numeriske beregninger. En må være varsom, slik at numeriske feilberegninger ikke blir tolket fysisk og bidrar til misoppfatninger, slik Sørby og Angell (2012) observerte blant fysikkstudenter (kapittel 2.5).

Det finnes muligheter for å utvide undervisningsopplegget videre i dybden. Elever som vil ha en utfordring, kan for eksempel undersøke om det er mulig å ta høyde for at akselererte ladninger avgir stråling og implementere energitapet i koden.

OPPLEGG 3: Fotonkalkulator (Fysikk 1)

Dette undervisningsopplegget handler om å bruke programmeringsspråk til å lage et nyttig produkt – en brukervennlig kalkulator/maskin som kan ta inn verdier fra en bruker, gjøre noen nyttige beregninger og returnere disse. Her lager elevene «Fotonomatoren», en foton-automat som tar inn en bølgelengde og gir ut frekvens, energi og farge på fotonet. Opplegget passer for både Fysikk 1 og Fysikk 2.

TID: 2 timer (à 45 minutter)

UTSTYR:

- Datamaskiner med installert programvare Python/Anaconda

KOMPETANSEMÅL (FRA LÆREPLANEN I FYSIKK 1 OG FYSIKK 2)**:**

- definere og regne med begrepene frekvens, (…) bølgelengde og bølgefart (…)
- lage en eller flere matematiske modeller for sammenhenger mellom fysiske størrelser som er funnet eksperimentelt
- gjøre rede for Einsteins forklaring av fotoelektrisk effekt, og kvalitativt gjøre rede for hvordan (…) partiklers bølgenatur representerer et brudd med klassisk fysikk

LÆRINGSMÅL:

- Bruke programmering til å lage en kalkulator som håndterer brukerinntasting
- Regne mellom lysets bølgelengde, frekvens og fart og avgjøre lysets farge
- Lage og teste en algoritme med if–else-setninger.

STRUKTUR:

- *Oppstart*: Presentasjon av mål, programmering og algoritmisk tenkning
- *Felles eksempel*: Variabel-begrepet, brukerinntasting og betingelser (if-else)
- *Oppgave*: Lag en «fotonomator»-maskin

Undervisningen starter med at læreren presenterer målene for timen (Lysbilde I-3) og at læreren forklarer hvordan vi kan hente inn data fra brukeren av programmet og lagre det i en variabel (Lysbilde I-4). Læreren åpner Spyder og skriver et eksempel sammen med elevene. Eksempelet viser hvordan kommandoen `input` brukes, at inntastingen må konverteres fra en tekststreng til et desimaltall (float), og hvordan man kan regne med det lagrede tallet. Læreren viser hvordan elevene kan skrive ut en beskjed til skjermen, og hvordan man kan bruke `if-else`-setninger til å gi ulike tilbakemeldinger for ulike inntastede verdier.

Videre får elevene prøve seg selv, ved å lage «Fotonomatoren» (Lysbilde I-5). Brukeren av programmet blir bedt om å taste inn en bølgelengde, som elevene bruker til å regne ut frekvens og energi som skrives ut til skjermen. Basert på en liste over farger og tilhørende bølgelengder, skal elevene også skrive ut hvilken farge fotonet har. Programmet kan utvides ved at elevene også viser den aktuelle fargen på skjermen med en RGB (rød-grønn-blå)-kode, som også er oppgitt i tabellen. Elevene kan få en ekstra utfordring ved å utvide programmet slik at det viser glidende fargeoverganger ettersom bølgelengdene endres. Det kan gjøres ved å lage en lineær sammenheng mellom RGB-koden og bølgelengden i hvert av fargeintervallene.

I dette undervisningsopplegget arbeider elevene for det meste innenfor «digital problemløsning» i taksonomien over algoritmisk tenkning (Figur 2-3), tillegg til at det er noe «datahåndtering». Programmeringen har noe mindre nytte for å forstå selve fysikken i dette opplegget, men gir en alternativ tilnærming til å regne med formler og kan være med på å gi en visuell fremstilling av bølgelengder som farger. Dessuten legger dette grunnlaget for å lage mer avanserte kalkulatorer som elevene faktisk også kan anvendes i ettertid. I fremtiden kan elevene for eksempel bare åpne «fotonomatoren» og skrive inn bølgelengden når de skal regne ut energien i et foton. Dessuten er dette et eksempel på hvordan elever kan arbeide med entreprenørskap og lage produkter for et «eksternt publikum» (Pierson og Clark, 2018).

4.4 Utprøving og resultater fra klasserom

Metode

Undervisningsoppleggene er prøvd ut på elever ved en videregående skole i løpet av mars og april 2019. Observasjoner og tilhørende observasjonsnotater under gjennomføringen og en spørreundersøkelse besvart av elevene i slutten av undervisningen, utgjør datamaterialet for utprøvingen.

Til sammen er oppleggene testet ut tre ganger i et utvalg bestående av tre ulike klasser, med forskjellige lærere ved den samme videregående skolen i Norge. Lærerne har her fått navnene Birgitte, Camilla og Didrik. Av praktiske hensyn er dette et bekvemmelighetsutvalg (Robson og McCartan, 2016, ss. 280-281), og klassene er rekruttert fra den samme videregående skolen og med flere av de samme lærerne som også deltok i intervjuundersøkelsen (kapittel 3). Selv om klassene ikke er tilfeldig valgt, er dette nokså vilkårlige, typiske norske fysikklasser med nokså høye karakterer og noe overvekt av gutter. Det er dermed rimelig å anta at erfaringene fra disse tre klassene også vil ha overføringsverdi til andre fysikklasser.

Birgitte og Camilla deltok også i intervjuundersøkelsen om algoritmisk tenkning (se kapittel 3.1). Didrik har, som de andre lærerne, høyere utdanning innen fysikk. Han har mange års erfaring med undervisning – de siste årene hovedsakelig i matematikk og naturfag, men underviser nå også i fysikk. Alle lærerne har selv erfaring med programmering. Camilla har også noe erfaring med å undervise programmering for elever. Det har ikke Birgitte og Didrik. Didrik tar imidlertid flere kurs i programmering som etterutdanning i forbindelse med fagfornyelsen, og i likhet med de andre lærerne er han interessert i å prøve det ut blant elever.

Opplegg 1 om rettlinjet bevegelse ble gjennomført to ganger. Første gang på en fagdag i Fysikk 1 med elever fra studiespesialiserende utdanningsprogram i klassen til Birgitte, og andre gang på en fagdag i Fysikk 1 med elever fra idrettsfaglig utdanningsprogram i klassen til Didrik. Opplegg 2 om bevegelse i elektrisk felt ble gjennomført i en dobbeltime i Fysikk 2 i klassen til Camilla. Da opplegg 1 viste seg å ta lengre tid enn først antatt, ble opplegg 3 ikke gjennomført for en hel klasse. En del elever i klassen til Didrik ble imidlertid tidlig ferdige og fortsatte da med oppgavene i det tredje opplegget etter en kort introduksjon fra læreren.

Alle oppleggene ble gjennomført av forfatteren. Lærerne fikk tilbud om å undervise oppleggene selv, men foretrakk at jeg stod for gjennomføringen. Datainnsamlingen bestod

hovedsakelig av ustrukturert, deskriptiv observasjon og tilhørende observasjonsnotater (Robson og McCartan, 2016, s. 328). Som Robson og McCartan (2016, s. 321) påpeker, er observasjon i en ustrukturert form en godt egnet metode for å skaffe oversikt i en tidlig fase. Observasjon gjør det mulig å studere klassen som helhet, uten større inngripen i form av lyd- eller videoopptak. Det gjør det også praktisk enklere å gjennomføre, på bekostning av at det blir vanskeligere å gjøre en helt systematisk analyse.

Selv foretok jeg det Bjørndal (2013, s. 32) kaller «observasjon av andre orden»: at jeg som lærer observerte en pedagogisk situasjon jeg selv var en del av. Observasjonen ble dermed en sekundær aktivitet, ved siden av undervisningen. Det kan være utfordrende for læreren som selv gjennomfører undervisningen å ha fullstendig oversikt og tid til å notere sine observasjoner, da han gjerne blir opptatt av egen undervisning og av å hjelpe elevene. Jeg noterte mine erfaringer til dels undervegs, men hovedsakelig umiddelbart etter timens slutt. Som Bjørndal (2013) påpeker, vil mye da være glemt. For å sikre gode data ble derfor undervisningen også observert av en person som selv ikke underviste, av første orden. Under det første opplegget var det klassens lærer, henholdsvis Birgitte og Didrik, som observerte. Under det andre opplegget var klassens lærer, Camilla, ikke til stede, så en lektorstudent ved NTNU bidro som observatør. Observatørene skrev notater undervegs i timene. De fikk en innføring om algoritmisk tenkning og oppleggenes innhold på forhånd, samt instruksjoner om å observere når nye ord ble innført av læreren, innspill, forslag, spørsmål, frustrasjoner og utsagn fra elevene, samt eventuelle tegn på at elevene arbeidet i adskilte moduser (se kapittel 2.3). Observatørene ble imidlertid bedt om å ha et vidt fokus. Bjørndal (2013, s. 51) peker på at en slik fleksibel forskningsdesign kan være nyttig for å bli klar over interessante forhold som ellers ikke var forventet på forhånd, og en kan bli oppmerksom på sider ved undervisningen som bør ses nærmere på. Observatørene gikk også noe rundt i klasserommet og snakket med elevene, spurte hvordan det gikk og ba elevene forklare hva de gjorde eller hvordan de forstod ulike begreper. Selv om de hadde anledning til å støtte elevene som ventet på hjelp, var de i mindre grad der for å hjelpe elevene, slik at det ikke skulle bli mer lærerstøtte enn det normalt ville vært.

Observatøren siler, bevisst eller ubevisst, ut enorme mengder informasjon. Hva som observeres, og hvordan informasjonen tolkes i persepsjonsprosessen, påvirkes av observatørens forventninger og motivasjon (Bjørndal, 2013, s. 34). At observatørenes fokus her ikke var mer avgrenset, kan ha medført at viktig informasjon ble oversett og at observasjonene ble noe mer preget av tilfeldighet (Bjørndal, 2013, s. 51). Selektiv oppmerksomhet og hukommelse kan føre til forventningsskjevhet (Robson og McCartan, 2016, s. 331). Det er naturlig at den som har

laget undervisningsopplegget ønsker at det skal fungere, og dermed blir kanskje de positive sidene ved undervisningen tillagt ekstra oppmerksomhet. Det kan også ha omvendt effekt, at observatøren blir for selvkritisk. At undervisningen ble fulgt av flere personer i klasserommet, og med ulike personer i de ulike klassene, fungerte også som en «observasjonstriangulering». Det kan bidra til å redusere forventningsskjevheten og styrke troverdigheten og gyldigheten av den fleksible forskningsdesignen (Robson og McCartan, 2016, s. 171).

Det er ikke gjort noen formell analyse av observasjonsnotatene. Erfaringene som presenteres er gjort på bakgrunn av en sammenfatting av observasjonene, med formål om å beskrive hvordan undervisningsoppleggene utspilte seg i klasserommet og å peke på elevenes og lærernes opplevelser av dem. Jeg trekker frem sider ved undervisningen som kan karakteriseres som suksessfulle, og peker på utfordringer og hvordan oppleggene kan forbedres.

I slutten av undervisningen ble elevene bedt om å fylle ut et spørreskjema om deres tidligere erfaring med programmering, hva de opplevde utfordrende og interessant, og hvordan de opplevde innholdet i undervisningen (se vedlegg C). Besvarelsene fra spørreundersøkelsene ble analysert med frekvensfordelinger og grafiske representasjoner, samt enkel deskriptiv statistikk, som sentralmål og spredningsmål (Robson og McCartan, 2016, ss. 416, 418). Resultatene fra undersøkelsen gir noe mer kvantitative og objektive data, i tillegg til observasjonsnotatene fra lærerne i klasserommet. Undersøkelsene kan underbygge observasjonene, og eventuelt tilføye noe mer om elevenes opplevelse av timene. Spørreundersøkelsene fungerer dermed også som en metodetriangulering (Robson og McCartan, 2016, s. 171).

På grunn av utprøvingenes begrensede omfang, ble det ikke gjort lyd- eller videoopptak. Elevsitater og dialog som er gjengitt her, er basert på observatørenes notater og elevenes kommentarer i spørreundersøkelsene. Noen av sitatene er dermed gjengitt etter observatørenes tolkning og husk, og de kan avvike noe fra det som faktisk ble sagt av elevene. Elevene er nummerert alfabetisk med bokstavene A, B, C, …, for enklere å kunne henvise til bestemte elever og deres utsagn. I den grad det fremgår av observasjonsnotatene og har betydning for sammenhengen, har eleven beholdt samme bokstav. Det er en mulighet for at samme elev omtales som for eksempel både Elev A og Elev G, hvis observasjonsnotatene ikke har understreket at det er samme elev. I en fremtidig undersøkelse kunne det vært interessant å gjøre skjerm- og lydopptak mens elevene arbeider med programmering i fysikk for å analysere elevenes arbeid og diskusjoner grundigere, samt å kartlegge elevenes læringsutbytte.

Erfaringer og resultater fra opplegg 1

Opplegg 1 var det første opplegget som ble prøvd ut på elever, på en fagdag i Fysikk 1-klassen til Birgitte, og også det siste opplegget som ble prøvd ut, på en fagdag i Fysikk 1-klassen til Didrik. Her presenteres noen av de erfaringene som ble gjort og resultatene av observasjonene og spørreundersøkelsene. Jeg tar først for meg erfaringene som ble gjort i Birgittes klasse, før jeg gjør rede for noen av de justeringene som ble gjort før gjennomføringen i Didrik sin klasse, og til sist erfaringene og resultatene fra denne klassen.

Elevene får til mye med en del støtte

Erfaringen vi gjorde i Birgitte sin klasse, var at de aller fleste elevene jobbet godt og virket interesserte i arbeidet med programmering. Med en del støtte klarte de fleste å løse alle oppgavene. De atten elevene trengte en god del støtte, med å lagre filer på riktig sted, små skrive- og syntaksfeil i koden eller mer logiske og algoritmiske feil, som å hente riktige data fra riktig kolonne i datafilen eller å bruke riktig antall iterasjoner i en løkke. Elevene ble klar over hvor lite som skal til for å få feil, og en av elevene oppga også dette som det mest interessante ved timen i etterundersøkelsen. Vanlige feil var å forveksle store og små bokstaver, innrykk på feil sted eller gal tegnsetting. En elev med dysleksi fikk problemer fordi hun hadde skrevet variabelen «akselerasjon» på tre ulike måter, og dermed gjenkjente ikke Python den riktige variabelen. Det var mange spørsmål, men ettersom de fleste elevene samarbeidet i mindre grupper, og elevene uoppfordret spurte og hjalp hverandre, gikk det fint for læreren å hjelpe alle forholdsvis raskt.

Kodingen opplevdes nok som utfordrende for mange elever, som gjorde at noen også gav opp til tider. Erfaringen var imidlertid at de fleste ble mer interessert etter hvert som vi klarte å finne ut av problemene deres. En elev hadde fått feilmelding. Da sa medeleven: «Det sammen skjedde med meg. Jeg bare ga opp. Skjønte ikke hva som var feil.» Den første eleven fortsatte imidlertid å prøve, og etter litt hjelp fra læreren, fikk han det til, og han virket veldig fornøyd etterpå, ifølge læreren Birgitte. Da delte han sine erfaringer med klassekameratene. De aller fleste klarte å løse oppgavene til slutt, med en god del støtte fra læreren og andre medelever.

Programvaren bør lastes ned på forhånd

Elevene hadde ikke rukket å laste ned programvaren i forkant av undervisningen, så vi måtte bruke den første timen på å laste ned og installere. Anaconda er et stort program, og selv etter en time var det flere elever som ikke hadde klart å få programmet til å kjøre. På grunn av noe begrenset med tid, klarte vi ikke å finne ut av disse problemene, og valgte at disse elevene i

stedet skulle samarbeide med elever som hadde klart å installere programvaren. Erfaringen var at elevene som ikke hadde klart å laste ned programvaren, var mindre delaktige i timene. Det var unaturlig for dem å begynne å skrive på medelevens datamaskin, og dermed ble det en lang økt hvor de bare så på medeleven skrive. En av disse elevene forlot klasserommet omtrent midt i timen og sa til læreren: «Jeg liker ikke programmering. Det er ikke så gøy. Det er for avansert.» Dette illustrerer at det er viktig at elevene får skrive egen kode på egen maskin, selv om de samarbeider. Det var en noe uheldig start å måtte bruke så mye tid på nedlastning, ettersom en del elever aldri kom ordentlig i gang, mens andre elever begynte å kjede seg før undervisningen egentlig hadde begynt.

Mens elevene lastet ned, ble de presentert for hensikten og læringsmålene for timen, og de ble introdusert for begrepet «algoritmisk tenkning». Birgitte observerte at «elevene flakket med blikket, satt urolig og snudde seg mye», og det så ut til at elevene hadde problemer med å holde fokus. Det skyldes kanskje særlig at de også var opptatt av å laste ned programmet på datamaskinene sine samtidig.

For at elevene ikke skulle sitte og vente en hel time på at programmet ble lastet ned, valgte vi å bytte litt om på rekkefølgen i undervisningsopplegget, slik at elevene først gjorde den praktiske delen som omhandlet Tracker og innsamling av egne data i mellomtiden. Ulempen med dette var nok at elevene hadde en litt mindre forståelse for hvorfor de skulle filme bevegelsen og hva de skulle bruke den til, enn hvis de hadde fått startet med kodeeksemplene og Warholms 400-meterløp først. Dessuten ble det en mye lengre, uavbrutt økt med programmering i etterkant.

Tracker vekker interesse

Etter en demonstrasjon av Tracker-programmet, filmet elevene en egen rettlinjet bevegelse og analyserte den. Birgitte observerte at elevene nå var mer påkoblet. En gruppe diskuterte hvordan de skulle plassere linjalen i bildet i forhold til bilbanen. En annen gruppe diskuterte hvilket lodd de skulle bruke, og konkluderte med at de ville bruke «et annet enn de andre i alle fall, så det blir litt annerledes grafer».

Erfaringen er positiv med at elevene får filme sine egne bevegelser og analysere dem i programmeringsspråket. De får eierskap til den fysiske bevegelsen og ser den for seg når de studerer den grafiske representasjonen. Når elevene diskuterer grafens utseende, knytter de argumentasjonen til den fysiske bevegelsen.

Birgitte har i etterkant allerede tatt i bruk Tracker i undervisningen på egen hånd. Da erfarte hun at noen av elevene selv valgte å eksportere dataene og analysere dem med programmering.

Mange tekniske operasjoner

I løpet av opplegget blir det foretatt en del operasjoner, og i oppstarten blir det mange på kort tid: Installering av Anaconda, åpning av Spyder, opprette nye filer, lagre fil i en mappe, laste ned datafil i samme mappe, legge inn og analysere video i Tracker og å eksportere data fra Tracker i et lesbart format med punktum som desimaltegn. I tillegg er det mange nye begreper i forbindelse med koden: variabler, funksjoner, kommandoer, løkker, eksportere, importere… Dette var nok en litt brå oppstart for noen. Erfaringen var dessuten at mange av elevene hadde lite erfaring med en del grunnleggende håndtering av datamaskinen. Få klarte å installere den nødvendige programvaren på egen hånd, flere av elevene med Windows-maskiner hadde lastet ned Mac-versjonen, og overraskende mange elever trengte hjelp med helt enkle operasjoner som å lagre en fil eller laste ned en fil fra en nettside og flytte den til en mappe på datamaskinen sin. En av elevene hadde problemer med å skille begrepene «fil» og «mappe»:

> **Elev A:** Skal vi legge filen med tallene og koden i samme fil?

Det kan være et behov for å kartlegge elevenes forståelse for sentrale begreper i programmering og databehandling. Kanskje skyldes de manglende tekniske ferdighetene at disse elevene fra grunnskolen er vant til å arbeide med Chromebook-maskiner, hvor alt foregår i nettbaserte skytjenester uten det samme behovet for filhåndtering?

Samtidig er erfaringen at elevene får svært mange *ulike* problemer. Det kan for eksempel være programvare som ikke lar seg åpne på enkelte av elevenes datamaskiner, mangel på lagringsplass, feilmeldinger som er vanskelige å tyde, datafiler som får endret format og struktur ved nedlastning, eller at eleven har skrevet en «0» (tallet null) i stedet for en «o» (bokstaven o). Det skaper særlig utfordringer for læreren, som bør ha svært bred og solid IKT-kompetanse for å kunne løse problemene raskt. Det meste som kan gå galt, kommer til å gå galt. Selv om læreren kan å skrive kode, er det mange andre tekniske operasjoner han blir bedt om å håndtere.

Elevene ønsker å forstå mer av koden

På spørsmål fra Birgitte ganske tidlig i timen om hva de synes om programmering, svarte to elever, som har noe erfaring med programmering fra før:

> **Elev B:** Interessant. Vi har lært de ulike begrepene i matematikk og informatikk, som funksjoner, variabler osv.
> **Elev C:** Når du forstår alt, blir det litt morsommere.

På grunn av tidsbegrensningen var det ikke anledning til å gå i detalj på alle begrepene og tekniske detaljer knyttet til koden. Det ble lagt mer vekt på hva kodelinjene gjør, enn på å forklare alle kommandoer og kodesyntaks i detalj. Det opplevdes nok frustrerende for enkelte elever å ikke forstå fullstendig hva alt betydde.

> **Elev D:** Jeg hang ikke med, for det ble for mye. Ingen forstår noe som helst.

> **Elev E:** Tror ikke vi lærer så mye. Vi skriver bare av det han skriver. Skjønte egentlig ingenting. Jeg synes det er litt tungt. (…) Skjønner ikke hva vi skriver, det er litt fremmed.

Birgitte understreker at disse to elevene ikke har informatikk som valgfag, men at Elev E vanligvis har høy måloppnåelse i fysikk. Elevene uttrykker at de ikke ser vitsen med å skrive koden, for det lærer de ingenting av. Elev E sluttet derfor å jobbe. Elev D forsøkte å forstå hver linje, for det var ingen vits i å gjøre dette hvis han ikke forstod alt, men han syntes det var vanskelig. At elevene opplevde det som utfordrende å forstå selve koden, er også noe som gjenspeiles i resultatene fra spørreundersøkelsen, der åtte av elevene oppgir dette som det mest utfordrende.

Birgitte mener imidlertid at programmering godt kan være noe man lærer litt og litt, og at man bruker noe kode uten å forstå alle detaljer med en gang. Kanskje hadde elevene opplevd det som mer motiverende om de tydeligere hadde blitt forberedt på at de ikke skulle lære *å programmere* på fem timer, men at de skulle lære å *bruke* programmering som et *verktøy* i fysikk.

Feilsøking og feilretting

Vi observerte at elevene brukte flere ulike strategier for å lete etter og rette feil. På spørsmål om hvordan det gikk, svarte en elev: «Det går bra. Jeg har syntaks-error, da.» Denne forsøkte han å løse ved å søke på Internett.

En annen elev spurte sidemannen:

> **Elev F:** Hvordan fant du det?
> **Elev G:** Jeg satser på at det funker.

Denne eleven brukte altså en prøve–og–feile-strategi.

En annen gruppe, med seks elever, diskuterte resultatene sine ved å sammenlikne med hverandre. I denne gruppen deltok også Elev D aktivt, eleven som tidligere uttalte at «ingen forstår noe som helst».

> **Elev H:** Jeg fikk en lineær graf … La meg få se hva du har? Se her, du må definere tiden.
> **Elev I:** Jeg gjorde akkurat det samme, men jeg fikk ikke en rett linje.

Elevene diskuterer først at de har fått ulike resultater, selv om de har gjort akkurat det samme. Etter hvert blir de derimot enige om at de ikke kan ha gjort det samme, ettersom resultatet ble forskjellig, og da bestemte de seg for å sammenlikne linje for linje i koden.

Lek med problemløsning

Elev J og K arbeidet med den siste oppgaven, der elevene måtte tolke enhetene til de oppgitte dataene og bruke disse til å regne ut fart og strekning på lærerens løpetur. Utdraget er hentet fra deres diskusjon av enheter:

> **Elev J:** Fart blir jo minutter per kilometer … Blir det sekunder?
> (…)
> **Elev K:** Blir det 3? Det kan ikke stemme heller …
> (Rekker opp hånden og spør læreren Birgitte, som prøver å resonnere med dem)
> **Elev K:** Forresten, kan jeg bare få leke meg litte grann?
> (Elevene jobber videre)
> **Elev J:** Nå har du tegnet MANGE grafer!

Elev K ombestemmer seg og vil ikke ha hjelp fra læreren likevel. Han har lyst til å «leke» seg litt og prøve å finne en løsning på problemet selv. Strategien innebar blant annet å tegne «MANGE grafer». Eleven prøver altså å få en oversikt over datamaterialet ved å tegne grafer. Dessuten ser vi her hvordan den litt uvanlige formen på fartsdataene, målt i antall hele minutter

og sekunder per kilometer, la grunnlag for en diskusjon om enhetene til fart. Vi ser også at elevene bruker konkrete tallverdier for å vurdere om svaret kan være riktig.

Det var flere elever som hoppet over analysen av bevegelsen de hadde filmet selv, og gikk rett på den mer utfordrende siste oppgaven om lærerens løpetur. Flere så ut til å synes denne var interessant, fordi de skulle løse et problem og finne ut hvor lang løpeturen var. Det kan se ut til at når elevene har lært de grunnleggende kodeteknikkene, så synes de det er interessant å bryne seg på rikere problemløsningsoppgaver med programmering.

Elevene arbeider i dette opplegget med både programmering og algoritmer i det de forsøker å løse fysikkproblemer, men elevene får ikke en eksplisitt innføring i bruk av algoritmisk tenkning som en problemløsningsstrategi. Da Birgitte spurte noen elever om hva de trodde algoritmisk tenkning var, svarte de:

> **Elev L:** Grafer og sånn? At noe beveger seg og så lager man grafer som viser bevegelse.

> **Elev M:** Det er vel litt tallkoding og sånn. Jeg vet ikke helt hva algoritmisk tenkning er, men det virker ikke noe vanskelig.

I spørreundersøkelsen etter timen skrev en elev at han «lærte ganske mye i dag om programmering, men om jeg husker det er det en annen sak». Det er altså ikke gitt at elevene naturlig ser sammenhengen mellom programmering som et verktøy til å lage grafer og gjøre beregninger, og algoritmisk tenkning som en tankeprosess som kan brukes også uten datamaskinen til stede.

Resultater fra spørreundersøkelsen første gjennomføring (Birgittes klasse)

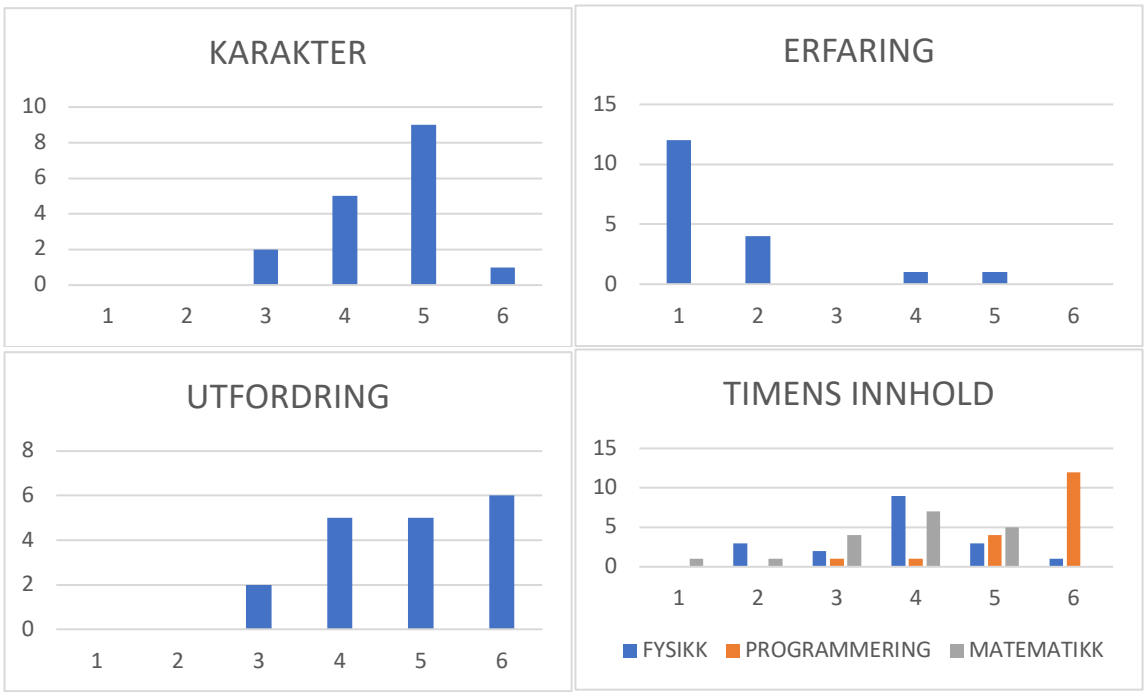
Elevenes svar på spørreundersøkelsen i slutten av undervisningsøkten er presentert i Figur 4-1. Den viser at elevene opplevde undervisningen som vanskelig. På en skala fra 1 (svært lite utfordrende) til 6 (svært utfordrende), er det ingen som velger de to laveste verdiene, mens 61 prosent velger de to høyeste verdiene, dette til tross for at de aller fleste elevene forventer karakterene 4 og 5 i standpunkt. Også de to elevene som oppgir at de har en del erfaring med programmering fra før, oppgir 4 og 5 som grad av utfordring.

Åtte av elevene oppgir at det mest utfordrende er å forstå selve koden i Python og logikken bak denne. En elev skriver for eksempel at det mest utfordrende var å «forstå når man skulle bruke de ulike kommandoene» mens en annen skriver «å vite nøyaktig hva man skal skrive i kodingen for å få alt rett». Flere av elevene oppgir at det mest utfordrende var at det ble

«altfor mye på en gang» og dermed «vanskelig å henge med». Tre elever synes det mest utfordrende var å «finne strekningen i oppgave 3».

På spørsmål om hva som var mest interessant, oppgir mange av elevene «å lære å kode» eller å «lage grafene». Et par likte best å filme og analysere sin egen bevegelse, mens fire svarer blankt eller er usikre. Mange av elevene oppgir at de lærte mye nytt om at «programmering kan brukes til mye mer enn jeg allerede trodde, forskning, beregninger og visualisering».

På spørsmål om hva elevene opplevde timens innhold bestod av, svarte de aller fleste at det inneholdt mye eller svært mye programmering, men mengden fysikk- og matematikkinnhold er også nokså høyt. Det kan tyde på at elevene opplevde at de faktisk arbeidet med fysikk, og ikke bare programmering. En av elevene skriver også at han «forstår nå bedre sammenhengen mellom strekning, fart og akselerasjon».



Figur 4-1 Resultater fra 18 besvarte spørreundersøkelser etter undervisning av opplegg 1 i studiespesialiseringsklassen. Elevene forventer høye karakterer i standpunkt, hadde ingen eller lite erfaring med programmering, og de opplevde undervisningen som utfordrende. De fleste syntes opplegget omhandlet mye programmering, men også at timene hadde høyt innhold av matematikk og fysikk.

Justeringer av opplegget

Etter gjennomføringen i Birgittes klasse ble det klart at opplegg 1 i all hovedsak fungerte godt, men at det også var behov for noen justeringer. Spesielt var det opprinnelige tidsestimatet på tre skoletimer litt optimistisk. Planen var å installere programvaren og også gjennomføre opplegg 3 i løpet av fagdagen på fem skoletimer. Det rakk vi ikke. Fire til fem timer på opplegg 1 uten installering av programvare er nok mer realistisk. Ved den andre gjennomføringen sørget vi for at elevene hadde programvaren installert på forhånd, slik at de fleste var klare til å starte med en gang. Vi satte av litt tid til å hjelpe de som hadde fått problemer, og vi antok at det bare ville bli tid til å gjennomføre opplegg 1.

Den første gjennomføringen ga også viktige erfaringer om hvilke feil elevene gjør og hva de opplever som utfordrende. Neste gang opplegget ble gjennomført, sørget jeg derfor for å ta meg ekstra god tid og vise stegene ekstra tydelig der jeg tidligere erfarte at elevene falt fra. Elevene ble også forberedt på at de ikke ville lære seg programmering fullstendig, og at de kanskje ikke ville forstå all koden med en gang, men at de skulle få erfaring med hvordan programmering kan brukes som verktøy i fysikk.

I Didrik sin klasse ble opplegget gjennomført i den rekkefølgen det egentlig var tenkt. Utover dette ble det ikke gjort noen vesentlige endringer i selve innholdet av opplegget.

Klassen til Didrik og klassen til Birgitte er forholdsvis like i kjønn- og karakterfordeling. Didriks klasse har elever fra studiespesialisering med idrett, mens Birgittes klasse har elever fra «vanlig» studieforberedende utdanningsprogram. På grunn av et idrettsarrangement, var en del elever fraværende da opplegget ble gjennomført i Didrik sin klasse. Dermed var gruppen noe mindre: 11 elever ved gjennomføringen i Didriks klasse, mot 18 elever i Birgittes klasse.

Tilsvarende erfaringer i Didriks klasse

Gjennomføringen i Didrik sin klasse underbygger mange av erfaringene som ble gjort i Birgittes klasse. Selv om elevene ble bedt om å installere programvaren på forhånd, var det nødvendig å bruke noe tid på å hjelpe en del elever i gang. Det gikk imidlertid mye raskere å hjelpe de få som hadde problemer, og det gjorde at vi kom mye tidligere i gang med programmeringen. Elevene jobber godt med programmering, og Didrik sier han ble «overrasket over hvor godt de fikk det til». Elevene trenger en del støtte fra læreren, men også her ser vi at elevene hjelper hverandre og deler løsningene videre. Også denne gangen så vi eksempler på at elevene ikke ville få for mye hjelp fra læreren: «Ikke si svaret, jeg vil finne det ut selv,» sa en elev til Didrik. En hjelpe-strategi som ser ut til å fungere godt, er å oppfordre elevene til å løse det samme

problemet for bare noen få konkrete tallverdier, og så be elevene gjenta den samme utregningen for noen få nye tallverdier, før elevene til slutt klarer å se mønsteret og lage en generell algoritme.

Elevene i klassen til Didrik var merkbart mer interessert i eksempelet om Karsten Warholm enn det elevene i klassen til Birgitte var, der bare et fåtall i det hele tatt hadde hørt om friidrettsutøveren. Kanskje var idrettseksempelet spesielt godt egnet for en idrettsklasse. Idrettselevene valgte også ut flere varierte rettlinjede bevegelser de skulle filme og analysere med Tracker. Noen så på fritt fall, noen på en trillende ball, noen så på en medelevs hopp, og en av elevene ville måle akselerasjonen på bilen sin, og tok med seg kompisene ut for å filme på parkeringsplassen.

Opplegget bidrar til gode diskusjoner

Noe vi også kunne observere i Birgitte sin klasse, men som Didrik spesielt trekker frem som en styrke ved undervisningsopplegget, er at det bidrar til gode fysikk-diskusjoner blant elevene. Et eksempel er når Elev N og Elev O diskuterer den «hakkete» grafen de har fått etter å ha beregnet akselerasjonen til Warholm numerisk.

> **Elev N:** Dette ser jo riktig ut, men det skulle vært mer jevnt.
> **Elev O:** Ja, men det er jo riktig. Det er jo ikke det at han stopper og løper bakover, det er jo bare at farten minker.
> **Elev N:** Jo'a …
> **Elev O:** Men endringen er jo bare 0,1.

Elev N hadde forventet en mer jevn akselerasjonsgraf, og Elev O forklarer hvorfor grafen likevel kan være riktig ved å gi en fysisk tolkning av akselerasjon. Elev N er enig, men likevel skeptisk, og Elev O understreker at endringene er veldig små, «bare 0,1». Ingen av elevene kommenterer her at den ujevne formen og de små utslagene på 0,1 antakelig skyldes feil i den numeriske beregningen. Læreren presenterer dette i den felles klassediskusjonen noe senere.

Didrik har observert flere tilsvarende diskusjoner, der elevene forsøker å koble grafene til fysikk-innhold. Han tror at en del elever ville godtatt alle grafer hvis de hadde kommet ut av et ferdiglaget program, for programmet har som regel rett. Her er elevene derimot grunnleggende mer skeptiske til resultatene, fordi de er usikre på egne programmeringsferdigheter. Elevene studerer grafene sine og konkluderer med at «den her posisjonsgrafen kan ikke stemme» og begynner å undersøke hva som kan ha gått galt og å tolke feilene. Eleven som analyserte bilen sin, oppdaget for eksempel at bilen startet ved -7,5 meter,

og kunne ikke forstå hvordan det kunne stemme. Det la grunnlag for gode diskusjoner med læreren og medelevene om valg av koordinatsystemets nullpunkt.

Programmering kan appellere til andre elever

En del av elevene arbeidet svært flittig med kodene og så ut til å synes det var interessant, men ifølge Birgitte var ikke dette nødvendigvis elevene med best karakterer i faget. Tvert imot kunne det se ut som om noen av elevene som ellers arbeidet svært godt i timene, heller gjorde lite. Birgitte hevder noen av elevene er svært karakterfokuserte, og ettersom programmering foreløpig ikke teller på karakteren, kan de være mindre motivert for å lære seg dette.

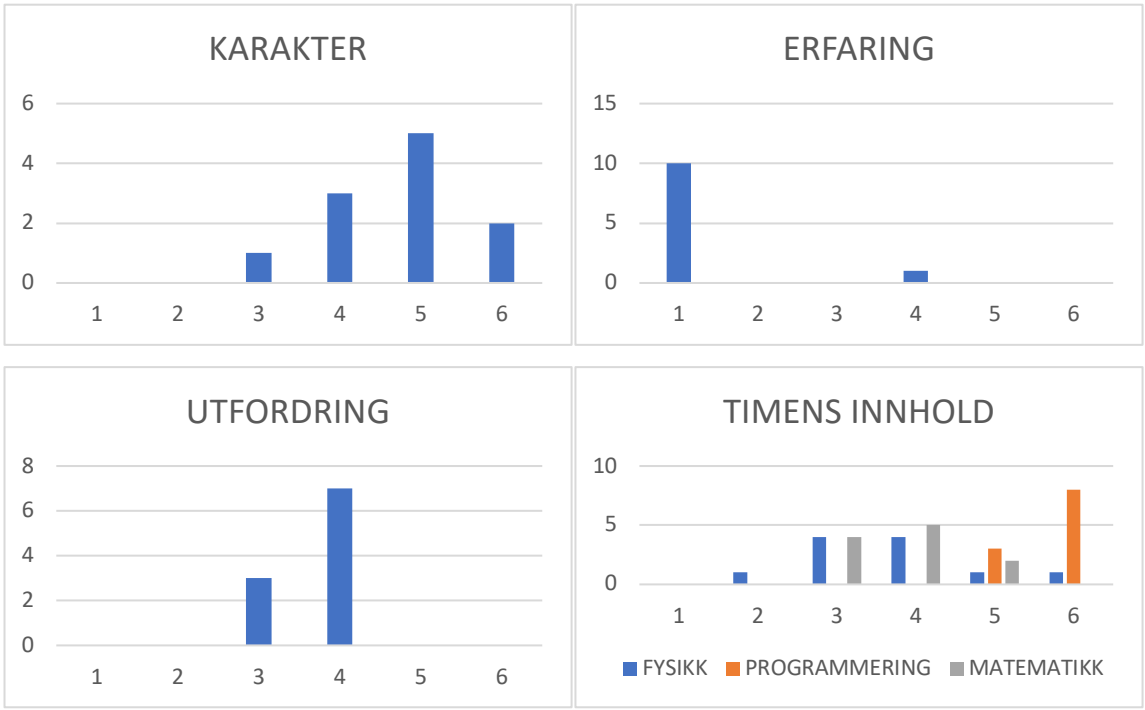
Dette fremkom også av spørreundersøkelsen. En elev som selv forventer karakteren 5 i standpunkt, svarer i undersøkelsen at «alt» var mest utfordrende og at «ingenting» var mest interessant med undervisningen. Totalt sju andre elever, fordelt på Birgitte og Camilla sin klasse, som alle også forventer femmere, svarer «ikke sikker», «vet ikke» eller blankt på spørsmålet om hva som var mest interessant.

Dette samme ble observert i Didriks klasse. Det han omtaler som «et arbeidsjern» i vanlige timer, var «mer slapp» denne økta, og han regnet med at det skyldtes at programmering ikke påvirket karakteren. Motsatt var det flere av elevene som ifølge Didrik vanligvis oppnår middels gode resultater, som ble svært engasjerte. En av elevene ble sittende alene inne i friminuttet og fortsatte å prøve å rette en feil han ikke klarte å løse. «Det hadde aldri skjedd med en vanlig oppgave i boka», forteller Didrik i etterkant. Andre elever søkte opp Python på Internett for å undersøke hvilke andre muligheter som finnes med programmering, og noen forsøkte på egen hånd å kopiere kode for å lage små spill og animasjoner.

> **Elev P:** Det her var jo kjempekult!

I etterkant ble jeg kontaktet av elevenes matematikk-lærer, som hadde fått høre om programmeringen i fysikk fra elevene. Elevene hadde spurt om de ikke kunne få bruke programmering i matematikk-timen også, og på «del 2» av tentamen og eksamen.

Dette viser at det ikke nødvendigvis er elevene med høyest måloppnåelse i fysikk som var mest engasjerte i undervisningen med programmering. Det er altså mulig at programmering appellerer til andre elevgrupper enn tradisjonell undervisning. Liknende erfaringer har blitt gjort tidligere, da for eksempel Aalmen (2010) undersøke drama som metode i undervisning av kvantefysikk.



Figur 4-2 Resultater fra 11 besvarte spørreundersøkelser etter undervisning av opplegg 1 i idrettsklassen. Elevene forventer høye karakterer i standpunkt, hadde for det meste ingen erfaring med programmering, og de opplevde undervisningen som middels utfordrende. De fleste syntes opplegget omhandlet mye programmering, men også at timene hadde middels innhold av matematikk og fysikk.

Resultater fra spørreundersøkelsen andre gjennomføring (Didriks klasse)

Figur 4-2 viser resultatene fra spørreundersøkelsen i klassen til Didrik etter gjennomføringen av opplegg 1. Den viser at elevene i Didriks klasse forventer omtrent samme karakterfordeling som klassen til Birgitte, har like lite erfaring med programmering fra tidligere, og opplever fremdeles at timen inneholder mye programmering, men også en god del fysikk og matematikk. Elevene i Didriks klasse opplever derimot ikke timen som svært utfordrende, slik elevene i Birgittes klasse gjorde. Samtlige elever har her svart middels vanskelig. Det er særlig interessant, ettersom timens innhold og elevgruppen i all hovedsak er lik. At elevene her opplevde opplegget som mer passe utfordrende kan kanskje skyldes de mindre justeringene i hvordan innholdet ble presentert, at alle var ferdige med installeringen før gjennomgangen, at elevene ble forberedt på at de ikke nødvendigvis ville forstå all kode, og at de fikk tettere oppfølging som følge av at elevgruppen var noe mindre.

Samtlige elever oppgir at det mest utfordrende var å forstå programmeringsspråket, vite hvilke kommandoer de skulle bruke, og å få programmet til å gjøre det de ønsket.

På spørsmål om hva som var mest interessant, svarer elevene blant annet: «å programmere», at «alt var matte», at «det var morsomt å se alt man kunne gjøre med programmering, når jeg googlet på google. Det var morsomt å gjøre oppgave 2, hvor man brukte Tracker + Python», «å få ut resultater som man kan sette i sammenheng med det virkelige livet. Praktiske oppgaver med strekning, fart og tid var derfor veldig bra for å forstå bedre» og «Det var interessant å se hvor bra og presist ting ble regnet ut og grafer lagd».

Flere elever oppgir også at de lærte mye nytt: «Det meste var nytt. Store tallmengder / data / målinger. Lærte mye nytt i dag; språk, kommandoer,» oppgir en elev, mens en annen skriver at «man kan bruke programmering til å løse oppgaver med mye data og lage diagrammer med disse dataene. Regne akselerasjon, fart, strekning osv. til simuleringer og elektriske felt. Lærte mye nytt om å bruke andre programmer enn GeoGebra osv. til å regne ut og lage diagrammer». Elevene opplever programmering som nyttig, og ser for seg at man kan bruke det i mange sammenhenger: «Mye man kan gjøre, blant annet å regne med kvanter, fart, tid, og mye annet. Tipper man kan gjøre det meste av fysikk der», «alt mulig», «programmering kan være lurt å kunne hvis man skal regne ut oppgaver med mange, mange målinger osv.»

Erfaringer og resultater fra opplegg 2

Etter erfaringene fra det første opplegget, sørget vi denne gangen for at elevene hadde lastet ned den nødvendige programvaren på forhånd. Dermed var stort sett alle med fra start. Opplevelsen var at disse elevene fulgte mer med under introduksjonen for opplegget, og det ble tydelige reaksjoner blant elevene da de ble fortalt om hvordan algoritmer kan brukes til overvåking. Elevene nikket også anerkjennende da de ble presentert for sammenlikningen mellom en funksjon i Python og en funksjon i matematikk. Det var imidlertid også her enkelte elever som opplevde at Anaconda ikke lot seg åpne på datamaskinene deres. Disse elevene ble derfor bedt om å samarbeide med sidemannen. Også denne gangen observerte vi at disse elevene var mindre delaktige i arbeidet.

Utfordrende å knytte sammen ulike fagområder

Elevene skulle først forsøke å regne ut avstanden mellom to punkter. Elevene ble oppfordret til å lage en figur og tenke logisk hvordan de kunne løse denne oppgaven. Selv om dette er et relativt enkelt matematisk problem, var det mange elever som stod fast. Elevene ble nok litt opphengt i programmeringsspråket, og de var usikre på hva de hadde lov til å gjøre. Når elevene derimot ble oppfordret til å tenke hvordan de hadde løst problemet uten programmering, var

det flere som kom frem til at de måtte bruke pytagorassetningen. Når elevene så skulle skrive inn dette i programmeringsspråket, var det flere som brukte variablene `x_0`, `x_1`, `y_0` og `y_1` til å beskrive koordinatene, slik de kanskje er vant med fra matematikk, selv om disse aldri var definert i koden. Dermed kjørte ikke koden. I funksjonen ble elevene nemlig bedt om å ta inn de to punktene som lister/vektorer, og de trengte hjelp til å bytte ut for eksempel `x_0` med `A[0]`. Det ble også observert at noen elever valgte å sette inn bestemte tallverdier i stedet for å regne generelt med variabler.

Elevene trengte på tilsvarende måte også hjelp med å tegne situasjonen når de skulle beregne kraftkomponentene, og de måtte minnes om definisjonene av cosinus og sinus. En av elevene visste at han skulle bruke trigonometri, men satt likevel fast.

> **Elev Q:** Men vi har jo ikke vinkelen?!
> **Lærer:** Men kan du regne ut cosinus og sinus uten å vite vinkelen, da?
> **Elev Q:** Hmm…

Etter en stund valgte læreren å skrive opp definisjonene med kateter og hypotenuser på tavlen, men også nå trengte noen elever hjelp med å se at hypotenusen er avstanden mellom punktene, og at katetene kan regnes ut ved hjelp av *x*- og *y*-koordinatene til punktene.

Utfordrende å generalisere

En sentral del av undervisningsopplegget er å lage ferdig funksjoner som generelt kan regne ut avstanden og kraften mellom to ladninger i to vilkårlige posisjoner. Det var nok uvant for elevene å generalisere utregningene og å tenke variabler og gjenbruk av funksjoner.

> **Elev R:** Hvis vi vet at avstanden mellom A og B er 5, kan vi ikke bare bytte ut det?

Utsagnet fra eleven, viser at han foretrekker å skrive inn tallet 5, i stedet for å regne med funksjonen `distance(A, B)`. Etter å ha blitt forklart hvorfor det er best å regne generelt, slik at funksjonen også kan fungere når det sendes inn andre punkter A og B, var eleven enig i at det var lurt.

> **Elev S:** Man vet jo aldri. All programmering er jo forskjellig.
> **Elev R:** Dette var jo vanskelig og artig, men viktig. Veldig viktig!

Fysikkinnhold

Vi observerte at elevene også diskuterte fysikkinnhold når de lagde funksjonene i programmet. I funksjonen for Coulombs lov, diskuterte de verdien av konstanten k og hva denne egentlig var. Elevene slo opp formelen for Coulombs lov og ulike konstanter i fysikkbøkene sine, og en elev som hadde lagt igjen læreboka si hjemme, kommenterte at «det er typisk at den eneste gangen man trenger boka, er når vi har programmering». Det kan se ut til at opplegget fikk elevene til å lete frem og friske opp hva de har lært før. To elever diskuterte hva et positron egentlig var, og kom frem til at det ikke var det samme som et proton, men antipartikkelen til et elektron.

De fleste elevene rakk akkurat å få ferdig koden slik at de kunne kjøre simuleringen i løpet av dobbeltimen. Det ble imidlertid ikke tid til å utforske simuleringen og diskutere de kvalitative observasjonene. Med et par timer til, ville det vært mulig for elevene å utforske de fysiske fenomenene mer, om hvordan felt, krefter, masser og ladninger påvirker bevegelsen.

Resultater fra spørreundersøkelsen (Camillas klasse)

Figur 4-3 viser resultatene fra spørreundersøkelsen i etterkant av undervisningen med opplegg 2. Elevene opplevde undervisningen som noe utfordrende, men denne gangen er det relativt færre som har valgt de to høyeste verdiene enn ved første gjennomføring av opplegg 1. Elevene oppgir at de forventer høye karakterer i standpunkt, og det er en noe større andel som har noe erfaring med programmering fra før.

Elevene er nokså samstemte når de blir bedt om å beskrive hva som var mest utfordrende: «Å skjønne hvordan man skulle sette opp programmet», «Å finne ut enkelte formler som skulle brukes i koden», «Skrive kommandoer slik at datamaskinen forstår hva den skal gjøre. Innebærer riktig tegnsetting osv.», «Det å vite kommandoene». Altså er det å forstå selve programmeringen og de ulike kommandoene mange opplever som mest utfordrende. En elev oppgir at det vanskeligste var at det var «mye nytt, en ny måte å tenke på», mens en annen skriver at det å «koble fysikken sammen med programmeringen var verst. Der må man tenke litt. Ellers enkelt for man må kun følge en oppskrift.»

På spørsmål om hva de synes er mest interessant, svarer flere det å kunne lage en animasjon og simulere bevegelse, og mange synes det er interessant å lære kode, lage sitt eget program og til å «få ut noe nyttig». En elev skriver at det mest interessante var å se «hvordan

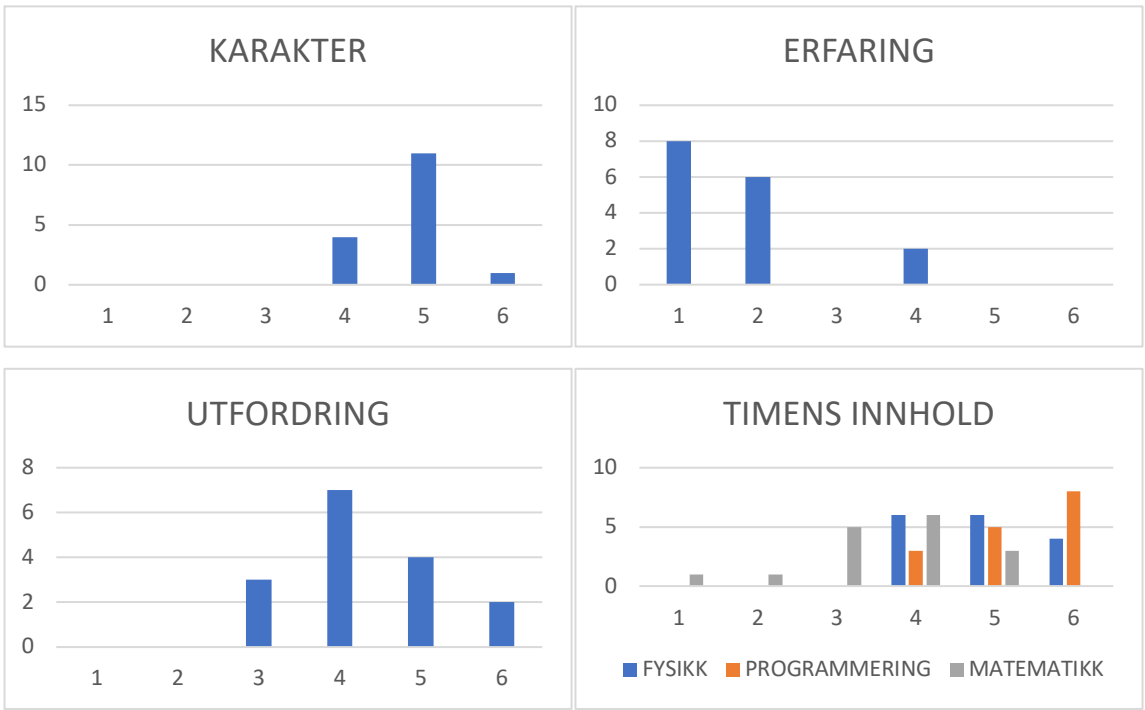
en liten kommando fører til utregning på sekundet», og en annen «de potensielle framtidsutsiktene for programmering». En elev likte «alt egentlig. Morsom time». Elevene svarer at de tror programmering kan være nyttig i simulering og beregninger, i ingeniørarbeid og roboter, i mekanikk og ellers til «alt».

Elevene beskriver innholdet i timen som bestående av mye programmering, men også mye fysikk og matematikk.

Erfaringer og resultater fra opplegg 3

Ettersom opplegg 1 trengte mer tid enn først antatt, ble det ikke mulighet til å gjennomføre opplegg 3 for en hel klasse. Noen av elevene i klassen til Didrik ble likevel tidlig ferdige med opplegg 1, og disse fikk da starte på opplegg 3 etter bare en kort introduksjon til if-setninger.

Erfaringen er at disse elevene klarte de første oppgavene greit. De klarte raskt å implementere brukerinntasting og å returnere passende utskrift. Kun én av elevene rakk å begynne på den mer utfordrende oppgaven om å vise graderte farger basert på inntastet bølgelengde. Med noen hint om å tenke linjer med stigningstall og konstantledd, klarte han å implementere den ene gradvise overgangen fra rød til gul i løpet av timen.



Figur 4-3 Resultater fra 16 besvarte undersøkelser etter undervisning av opplegg 2. Elevene forventer høye karakterer i standpunkt, hadde ingen eller litt erfaring med programmering, og de opplevde undervisningen som litt utfordrende. De fleste synes opplegget omhandlet mye programmering, men også at timene hadde høyt innhold av matematikk og fysikk.

4.5 Oppsummering

I dette kapitlet har jeg vist hvordan noen undervisningsopplegg i fysikk med algoritmisk tenkning og programmering kan se ut for elever uten programmeringserfaring, basert på kompetansemålene i dagens læreplan.

Eksemplene viser at numerisk beregning og programmering passer godt i arbeid med krefter og bevegelse, og at det kan være egnet til å lage «formelmaskiner»/kalkulatorer basert på enkle fysikkformler. Det er sannsynligvis også mulig å lage gode opplegg som knytter seg til andre former for datahåndtering med sensorer.

Det viser seg imidlertid å være utfordrende å finne gode numeriske eksempler fra andre områder i fysikken. Matematikken og programkoden blir fort utfordrende hvis elevene må utvide til flere dimensjoner. Når elevene om noen år har opparbeidet seg flere programmeringsferdigheter gjennom grunnskolen og matematikkfaget i videregående skole, vil det nok være noen flere oppgaver elevene kan klare å arbeide med. Samtidig handler mange av målene i dagens læreplan om kvalitativ forståelse og å gjøre rede for fenomener, som gjør det unaturlig å bruke mye tid på kvantitative og numeriske beregninger. Områder som kanskje er særlig egnet for numeriske beregninger, som kondensator-kretser, termisk og statistisk fysikk og mer avanserte elektromagnetiske eller kvantemekaniske beregninger, er i dag langt utenfor læreplanen i skolefysikken.

For at undervisningen skulle ha en del fysikkinnhold, ble det valgt å starte litt brått med nokså avansert programmering. Utprøvingene i klasserommet med elever fra både Fysikk 1 og Fysikk 2 viser at elevene likevel klarer å løse oppgavene og håndtere programmeringen, til tross for at de ikke har programmeringserfaring fra før. Undersøkelsene viser at selv om elevene opplever at timene inneholder svært mye programmering, så opplever de også at det er en god del fysikk- og matematikk-innhold. Det ser dermed ut til at programmering kan innføres raskt som et verktøy i fysikkundervisningen.

De første observasjonene og svarene fra spørreundersøkelsene tydet på at oppleggene hadde en litt for brå start, og elevene opplevde det som svært utfordrende å forstå koden. Med noen mindre justeringer i opplegget og at læreren ble bevisst hva elevene opplevede som utfordrende, tok seg god tid, og sørget for nok støtte, så det ut til at det samme opplegget kunne oppleves som passe utfordrende for elevene.

Når programmering innføres systematisk i skolen, vil det være naturlig å starte med noen timer med mer grunnleggende programmering, særlig i kombinasjon med matematikk-

undervisningen. Likevel kan man oppmuntre elevene til å bruke koden selv om de ikke forstår alt med en gang. Erfaringene fra utprøvingen viser at elevene kan bruke programmering til å undersøke fysikkfenomener, selv om de ikke forstår alle detaljene i koden. Det kan se ut til at denne måten å arbeide på, fungerer særlig godt og vekker interesse blant noen av elevene som ellers ikke er vant til å få toppkarakter.

Undervisningsoppleggene med programmering ser ut til å stimulere til gode fysikkdiskusjoner, og elevene forsøker å koble de grafiske representasjonene til de fysiske fenomenene. Det ser ut til at elevene er naturlig skeptiske til resultatene de får, og de begynner å vurdere hva de hadde forventet og hva som kan være mulige feil. Det kan se ut til at arbeidet med programmering gir elevene god trening i å generalisere og abstrahere formler, ved at de går fra konkrete talldata til generelle formler og algoritmer.

Under utprøvingene så vi flere eksempler på at elevene gjerne vil leke seg med programmering og prøve på egen hånd uten at læreren gir svaret. Elevene samarbeider uoppfordret med hverandre og deler løsningsstrategier. Samtidig viser utprøvingene at elevene ofte står fast og er avhengige av tett oppfølging fra læreren. Det meste som kan gå galt, går galt, og læreren må være forberedt på å håndtere både tekniske og logiske problemer. Det kan være utfordrende å gi nok oppfølging i større elevgrupper, men læreren får god hjelp av at elevene samarbeider og deler løsningsstrategiene sine med hverandre.

5 DISKUSJON OG KONKLUSJON

I de foregående kapitlene har jeg, gjennom en litteraturstudie, en intervjuundersøkelse og utvikling av undervisningsopplegg med utprøving blant fysikkelever, undersøkt hvordan programmering og algoritmisk tenkning kan og bør integreres i skolens fysikkfag. I dette kapitlet forsøker jeg å besvare problemstillingen gjennom en diskusjon, og jeg forsøker å trekke noen konklusjoner og peke på mulige implikasjoner for lærere, lærerutdannere, skoleforskere og læreplanforfattere.

5.1 Hva styrer fagfornyelsen mot?

Bakgrunnen for denne oppgaven er at programmering og algoritmisk tenkning er på vei inn i skolens læreplaner. Ikke som et eget fag, men integrert i skolens allerede eksisterende fag – spesielt i matematikk og naturfagene.

Det er for meg ikke helt klart hvordan vi endte med akkurat denne modellen for innføring av programmering i norsk skole. Selv om det finnes enkelte (diSessa, 2018; Malthe-Sørenssen et al., 2015; Papert, 1980; Wing, 2006) som taler varmt for innføring av algoritmisk tenkemåte som en grunnleggende ferdighet, har det etter hva jeg kan se, ikke vært noe sterkt ønske om å innføre dette fra hverken lærere eller forskningsmiljøet. Snarere tvert imot (Utdanningsdirektoratet, 2017a), og det ser ut til å være et resultat av politiske visjoner og økonomiske spørsmål.

Det finnes viktige samfunnsmessige og allmenndannende grunner for at flere bør lære seg teknologi og programmering, og det er viktig både demokratisk og økonomisk at folket forstår virkemåten til en datamaskin og algoritmenes plass i det tjueførste århundret (Bocconi et al., 2018). Jeg har likevel ikke funnet forskningsmessig belegg for at dette bør skje gjennom skolens matematikk- og fysikkundervisning. Når det er sagt, har jeg heller ikke funnet belegg for å si at det bør frarådes. Programmering har blitt et viktig arbeidsverktøy i fysikk, og det er naturlig at elevene lærer seg dette verktøyet og får et riktigere bilde av hvordan fysikere og ingeniører faktisk arbeider. Norge begir seg dermed, sammen med en rekke andre land, ut på et spennende eksperiment der det ikke er helt klart hva resultatene vil bli. Det er heller ikke helt klart om hovedmålet er å lære elevene programmering som verktøy, eller om programmering

skal inngå som en læringsstrategi for å styrke opplæringen i det innholdet som allerede utgjør dagens fysikkfag. Kan hende er det heller ingen motsetning mellom disse.

5.2 Begrepet algoritmisk tenkning

Utdanningsdirektoratet (2019a) har slått fast at begrepet «algoritmisk tenkning» skal forstås som en norsk oversettelse av begrepet «computational thinking». Det betyr at elevene ikke bare skal lære å programmere og å utvikle algoritmer, men også å bruke teknikker fra informasjonsteknologi som en generell problemløsningsstrategi.

Det viser seg at denne videre forståelsen av begrepet kan ha en naturlig plass i fysikkundervisningen, og spesielt Weintrop et al. (2016) har utviklet en taksonomi for hvordan dette kan se ut i praksis. Med kategoriene datahåndtering, modellering og simulering, digital problemløsning og systemtenkning, dekker de arbeidsmåtene som utgjør algoritmisk tenkning i matematikk og naturfag.

Intervjuundersøkelsen jeg gjorde med fysikklærere, viste imidlertid at «algoritmisk tenkning» som begrep, kan være en uheldig oversettelse av «computational thinking». Kanskje passer begrepet bedre i matematikk, hvor algoritmer tross alt kan ha en mer naturlig plass. I fysikk oppleves det imidlertid noe kunstig å snakke om algoritmer, og vi ser at fysikklærerne forbinder begrepet mer med en snevrere tolkning av algoritmelære, enn som en vidt anvendbar problemløsningsstrategi. Kanskje er det også derfor begrepet ikke forekommer i utkastene for nye læreplaner i naturfag, hvor det i stedet benyttes begreper som «teknologi» og «entreprenørskap» (Utdanningsdirektoratet, 2019b).

«Computational» blir i fysikksammenheng ved norske universiteter gjerne oversatt med «beregningsorientert», men som jeg påpekte i kapittel 2, er dette heller ikke et heldig begrep. «Computational thinking» handler ikke bare om å kunne utføre numeriske beregninger, selv om det er en naturlig anvendelse i fysikk. For at intensjonene bak «computational thinking» ikke skal reduseres til matematisk utvikling av algoritmer, er det viktig at lærerne får tilstrekkelig opplæring i begrepets innhold, og jeg har foreslått å se etter et alternativt begrep. Etter inspirasjon fra det svenske begrepet, kunne vi for eksempel brukt «datalogisk tenkning».

5.3 Vil programmering medføre algoritmisk tenkning?

I utprøvingen av undervisningsoppleggene observerte vi at en del elever lekte seg med programmering og problemløsning, og at enkelte elever ble motivert til å fortsette arbeidet og

lete etter feil også i friminuttene. Elevene undersøkte dessuten på Internett hvilke andre muligheter programmering kunne ha, de så for seg en rekke anvendelser i fysikkfaget, og de ønsket å få bruke programmering som hjelpemiddel på eksamen i andre fag.

Samtidig så vi også at en del elever kun knyttet programmering til det å tegne grafer, og det var lite som tydet på at disse elevene opplevde programmering og algoritmisk tenkning som en generell problemløsningsstrategi de kunne anvende også uten datamaskinen.

Hvis observasjonene til Sørby og Angell (2012) gjelder, om at studentene har problemer med å kombinere programmeringen med fysikken og arbeider i adskilte moduser, eller at den kognitive lasten blir for stor (Taub et al., 2015), kan vi risikere at opplæringen i programmering ikke bidrar til læring av fysikk. Elevene kan bli dyktige i programmering i seg selv, og kanskje også dyktige i å bruke programmering som et verktøy for å utføre beregninger i fysikk, men det er ikke gitt at dette medfører en algoritmisk tenkemåte når de skal løse fysikkproblemer i andre sammenhenger.

Det kan derfor være nødvendig at elevene også får opplæring i algoritmisk tenkemåte som en problemløsningsstrategi, og hjelp til å anvende denne systematisk. Det kan, som Brown og Wilson (2018) og Sørby (2010) har påpekt, gjøres gjennom modellæring, der for eksempel læreren eller oppgaveteksten fungerer som modeller. Visjonene til Wing (2006) vil sannsynligvis ikke kunne realiseres uten at elevene først behersker de grunnleggende programmeringsteknikkene. Elementene i algoritmisk tenkning ser ikke ut til å falle helt naturlig for elevene som ikke har lært programmering enda. Som vi så i utprøvingen av opplegg 2, var det utfordrende for elevene å generalisere og tenke gjenbruk av variabler og funksjoner. Fysikklæreren Camilla har gjort den samme erfaringen, og i intervjuet presiserer hun at «(…) elevene klarer ikke tenke sånn». Elevene må gjøres kjent med de grunnleggende teknikkene som kan anvendes i programmering før denne tenkemåten kan overføres til andre problemstillinger.

5.4 Trenger fysikkfaget en større overhaling?

Måten teknologi og programmering er tenkt innført i fysikkfaget, gir muligheter for å gjøre større endringer i fysikkfagets innhold, fordi elevene kan håndtere mer kompliserte problemstillinger med dette verktøyet. Hvis programmering skal ha en sentral plass i fysikkfaget, kan det være nødvendig å vurdere om det er andre temaer som er bedre egnet enn de som finnes i dagens læreplan.

Jeg har med tre undervisningsopplegg vist at det går an å knytte programmering til dagens læreplaner i Fysikk 1 og Fysikk 2, i alle fall om vi legger litt godvilje til, og er romslige i tolkningen av enkelte mål. Slik kan programmering brukes som et «krydder» til en ellers nokså tradisjonell tilnærming til faget.

Læreplanforfatterne må likevel vurdere hvilken historie vi ønsker å fortelle elevene. Som blant annet Malthe-Sørenssen et al. (2015) og diSessa (2018) har demonstrert, er det mulig å gjøre større endringer i form og innhold med en numerisk tilnærming til faget og programmering som en sentral teknikk. Ingeniørvitenskap og realistiske fysiske beregninger kan utgjøre hoveddelen av faget, på bekostning av unaturlige forenklinger og rent kvalitative forklaringsmodeller.

Jeg har ikke funnet noen evidens i forskningen som taler spesielt for det ene eller det andre. Ulempen med den tradisjonelle tilnærmingen er at elevene antakelig ikke får brukt programmering nok til at det er verdt tiden det tar å lære seg å programmere. Som Caballero et al. (2013) opplevde, vil elevene antakelig glemme hvordan de kan bruke det som et verktøy mellom hver gang. Dette samme poengterte enkelte elever etter min utprøving: De hadde lært mye programmering, men var ikke sikre på om de kom til å huske det. En fagdag med programmering er nødvendigvis ikke nok til å lære programmering skikkelig, men heller ikke tilstrekkelig for å få et fullstendig innblikk i programmering som verktøy i fysikk, eller til å utvikle algoritmisk tenkning som en problemløsningsstrategi.

Ulempen ved den numeriske tilnærmingen kan være at faget blir for kvantitativt, og slik lærer Camilla i mitt intervju frykter, for avansert og med mindre rom til kvalitativ forståelse. Fysikkfaget åpner i dag dessuten for undring over ulike fysiske fenomener og kan også ha filosofiske dimensjoner om eksistens, ontologi og epistemologi. Slike sider ved faget, som kan gjøre det inspirerende for flere elever, vil kunne gå tapt om faget får et for stort fokus på praktisk anvendelse og ingeniørvitenskap. Camilla er også bekymret for at elevene ikke vil ha tilstrekkelige programmeringsferdigheter fra før, og at grunnleggende programmerings-opplæring vil ta for mye tid fra fysikken. Undervisningsoppleggene jeg gjennomførte viser derimot at det er fullt mulig å be elevene bruke nokså avansert programmering, selv om elevene ikke har noen programmeringserfaring fra før. Det så ut til at oppleggene kunne bidra til å fremme fysikkfaglige diskusjoner og å utvide forståelsen av enkelte begreper. Det ser også ut til at programmering kan ha en naturlig plass i å knytte fysiske bevegelser og grafiske representasjoner tettere sammen, og kan på den måten bidra til å utvide mulighetene for modellering i fysikk, slik det blant annet er omtalt hos Angell, Kind og Henriksen (2008). Det er imidlertid en fare for at programmering kan gi kognitiv overbelastning, og vi så at spesielt

den første elevgruppen opplevde undervisningen som (svært) utfordrende. Samtidig kunne det se ut til at elever som vanligvis ikke får toppkarakterer, opplevde mestring og ble ekstra motivert av undervisningen med programmering.

Mange av kompetansemålene i dagens læreplaner for både Fysikk 1 og Fysikk 2, handler om å «gjøre rede for» eller å «beskrive». Dette er noe elevene kan gjøre også etter først å ha gjort beregninger eller laget simuleringer. For eksempel kan simuleringen elevene lagde i mitt opplegg for Fysikk 2, være grunnlag for å beskrive inhomogene felt rundt en partikkel. Det jeg opplevde som utfordrende i utviklingen av undervisningsoppleggene, var imidlertid å finne fysikkemner som egnet seg for beregning når elevene har liten programmeringserfaring. I likhet med mange av artiklene i litteraturstudien (se for eksempel Caballero et al., 2012; diSessa, 2018; Malthe-Sørenssen et al., 2015; Tellefsen, 2018), endte også jeg opp med å bruke tema fra mekanikk. Det var også her lærerne i intervjustudien så for seg at programmering kunne komme til sin rett. I opplegget for Fysikk 2, om elektriske ladninger, så vi at krefter og bevegelse kunne plasseres i en elektromagnetisk ramme. Selve prinsippet om kraft og bevegelse, er likevel mekanikk. De samme teknikkene, med numerisk derivasjon og integrasjon, kan nok også brukes i en rekke andre eksempler som er relevante for skolens fysikkfag. Beregning av planetbaner og skrått kast følger naturlig. En del av de samme teknikkene kan antakelig også benyttes i forbindelse med lyd, bilde og sensorteknologi. Kanskje er det også mulig å gjøre beregninger med Jordas magnetfelt, magnetisk fluks og induksjon. Faradays lov presenteres for eksempel i dagens Fysikk 2, men jeg frykter at det raskt blir for avansert når elevene må integrere over flater eller arbeide med felt i flere dimensjoner. På samme måte er det antakelig utelukket å innføre Schrödingerlikningen og komplekse tall for å gjøre numeriske beregninger i kvantefysikk. Foruten mekaniske bevegelser, kan jeg se for meg at termisk fysikk kan ha en del gode eksempler som er egnet for numerisk beregning på elevenes nivå. Her er det dessuten mulig å inkludere randomisering, for eksempel med «brownske bevegelser» og én- eller flerdimensjonal virrevandring. Jeg kan også se for meg noen mulige anvendelser innenfor optikk, elektriske kretser med kondensatorer og kanskje med reelle data i medisinsk fysikk. Selvfølgelig er det også mulig for elevene å bruke programmering til å styre elektriske kretser og lage roboter, sensorer eller andre teknologiske oppfinnelser. Selv om dette også benytter fysikk, hører mye av denne typen anvendelse etter min mening heller hjemme i faget Teknologi og forskningslære.

Forskningslitteraturen gir ingen klare veiledninger om hvordan programmering bør integreres i fysikkfaget, foruten at det finnes konkrete eksempler på enkelte områder innen for eksempel mekanikk der programmering kan ha en naturlig rolle (Malthe-Sørenssen et al., 2015)

og eksempler på misforståelser som kan avdekkes med programmering (Hutchins et al., 2018). For læreplangruppa blir dette antakelig et spørsmål om hva fysikk-faget skal være, heller enn om hva som gir best læring. Det er mulig å gjøre små justeringer i formuleringene av enkelte av dagens kompetansemål, slik at programmering kan brukes til å krydre undervisningen. Programmering kan fungere som et ledd i å oppnå dybdelæring, ved at fenomener som tidligere er studert kvalitativt eller algebraisk, nå også kan studeres numerisk. Flere av elevene i utprøvingen oppga for eksempel at de hadde fått en økt forståelse for bevegelse, fart og akselerasjon, etter å ha jobbet med opplegg 1. Det er også mulig å gjøre omfattende endringer i form og innhold, slik at fysikkfaget forteller en annen historie, der algoritmisk tenkning og numerikk legger grunnlaget for hva elevene skal lære. Kanskje er det også mulig med en kombinasjon, der for eksempel hovedområdet mekanikk gis noe mer plass med en større numerisk del. Elevene kan da arbeide med numerikk og programmering sammenhengende over en lengre periode, innenfor et tema der det har en naturlig anvendelse. De numeriske teknikkene herfra kan da i enkelte tilfeller trekkes over i de andre hovedområdene, slik jeg for eksempel gjorde med bevegelse i elektrisk felt i opplegg 2, men at de øvrige hovedområdene bevarer et mer kvalitativt fokus og en tradisjonell tilnærming.

5.5 Algoritmisk tenkning og programmering i praksis

Jeg har i utarbeidelsen av undervisningsoppleggene i denne studien i stor grad tatt utgangspunkt i taksonomien til Weintrop et al. (2016). Jeg opplever at dette er et godt verktøy når læreren skal planlegge undervisning med programmering og algoritmisk tenkning. Praksisene i taksonomien er enkle å overføre til praksiser i fysikklasserommet, og taksonomien hjelper læreren med å inkludere de mange sidene ved algoritmisk tenkning.

De to første kategoriene i Weintrops taksonomi, «datahåndtering» og «modellering og simulering», vil nok for mange fysikklærere være praksiser de allerede benytter i sin fysikkundervisning. Elevene gjør målinger og samler data i mindre forsøk, behandler, analyserer og visualiserer dem, gjerne med digitale verktøy som regneark eller graftegner. Dessuten er det vanlig å bruke digitale modeller og simuleringer i innlæringen av fenomener og begreper og til å utføre forsøk som ellers er vanskelige å gjennomføre i klasserommet. Vi kan dermed si at fysikklærerne på mange måter benytter seg av elementer fra algoritmisk tenkning i dag. Samtidig er min opplevelse at fysikklærerne kan mye programmering fra før, og de er interesserte i å prøve ut mulighetene det kan gi for fysikkfaget. Med programmering som verktøy, vil det bli mulig å arbeide med større og gjerne realistiske datasett, og elevene

kan også utvikle modellene og simuleringene selv. I tillegg kan elevene begynne å arbeide med de to neste nivåene i taksonomien: «digital problemløsning» og «systemtenkning». Som vi så i intervjuundersøkelsen, er særlig det siste begrepet utfordrende for lærerne å definere. Praksisene innenfor denne kategorien er kanskje ikke noe vi naturlig forbinder med fysikk og programmering, men som taksonomien hjelper oss med å inkludere.

Lærerne har likevel en utfordrende oppgave med å gjøre elevene til algoritmiske tenkere. Det er ikke mulig å be elevene tenke algoritmisk, før de har fått en innføring i programmering og kjenner de grunnleggende strukturene i et program. Det er krevende å utforme undervisningsopplegg som gir en god innføring i programmering, uten å gå på bekostning av fysikkinnholdet. I undervisningsoppleggene som ble presentert i dette arbeidet, har jeg forsøkt å balansere mengden eksempler og instruksjon med elevenes mulighet til å fikle, utforske og arbeide selvstendig og sammen med problemløsning, slik Utdanningsdirektoratet (2019a) beskriver sentrale arbeidsmåter for en algoritmisk tenker. Elevene som oppleggene ble prøvd ut på, hadde hovedsakelig ingen erfaring med programmering fra før. Jeg valgte derfor å bruke demonstrerte eksempler som utgangspunkt, med læreren som modell, og oppgaver med presise instruksjoner i starten – etter inspirasjon fra blant annet Brown og Wilson (2018) og Sørby (2010). Dette er likevel ikke et hinder for flere oppgaver med en mer utforskende karakter og dybdelæring. Etter hvert blir oppgavene mer åpne og krever at elevene benytter seg av problemløsningsstrategier. Oppgavene får dermed, med terminologien til Sengupta et al. (2013), en nokså lav terskel, men et høyt tak. I utprøvingen hadde elevene behov for mye støtte, fra læreren og fra medelever, for å klare å løse de mest åpne oppgavene. Det er naturlig at de trenger mer trening i programmering og algoritmisk tenkning, før de kan bli mer selvstendige.

Det er relativt enkelt å finne måter å utvide problemene, slik at de får et høyere tak og lar elevene utforske videre i dybden. I opplegg 2 har jeg for eksempel vist hvordan elevene kan bli bedt om å utvide den første, enkle simuleringen med flere partikler, eller til og med å utvide koden slik at den tar høyde for stråling fra akselererte ladninger. Programmering gjør det mulig å utforske et fenomen grundigere, på måter som ikke lot seg gjøre med rent kvalitative og algebraiske teknikker. Det motsatte er derimot heller vanskelig. Når inngangsterskelen blir lav, er det noe mer begrenset med fysiske problemstillinger som lar seg utforske.

Mange av elevene som deltok i utprøvingen av mine undervisningsopplegg, oppga i undersøkelsen at det var svært utfordrende. Det kan tyde på at introduksjonen til programmering var litt for rask. Om disse elevene hadde mestret de grunnleggende programmeringsteknikkene på forhånd, ville antakelig grad av utfordring opplevdes noe mindre. Fra litteraturstudien har vi sett at å knytte sammen ulike fagområder er utfordrende for

elevene (se for eksempel Sørby og Angell, 2012). Det samme ble observert i utprøvingen, der for eksempel elevene i Fysikk 2 strevde med å bruke pytagorassetningen og enkel trigonometri i sammenheng med fysikk og programmering.

Likevel er min erfaring at det går fint å innføre programmering gradvis i kombinasjon med fysikken, og det er ikke nødvendigvis slik at elevene bør ha et separat programmeringskurs før de kan benytte det som verktøy, slik blant annet Malthe-Sørenssen et al. (2015) mener. Det krever imidlertid spesielt godt tilpassede undervisningsopplegg. Elevene i mine utprøvinger hadde forøvrig lært om og arbeidet med fysikkinnholdet før programmeringen ble innført. Den kognitive overbelastningen hadde sannsynligvis blitt for stor om både programmeringsverktøyet og fysikkbegrepene var helt ukjente og innført samtidig.

Elevenes svar på spørreundersøkelsen viser at de opplevde høyt fysikk- og matematikkinnhold i undervisningen – samtidig som de opplevde at programmering var hovedfokuset. Det trengs imidlertid mer forskning for å si noe om elevenes læringsutbytte og om hvorvidt programmering påvirker elevenes forståelse av fysikkbegreper og fenomener. Vi så også at en del elever hadde problemer med grunnleggende håndtering av datamaskinen, og det kan være nødvendig å kartlegge og forbedre elevenes dataferdigheter.

5.6 Studiens styrker og svakheter

Dette masterarbeidet har tatt for seg et enda nokså nytt og lite utforsket område. En av studiens store styrker, er da også at den gjennom ulike metoder forsøker å gi et overblikk over hvilken rolle algoritmisk tenkning og programmering kan spille i fysikkundervisning, og den baner vei for videre forskning og innføring i skolen.

De tre delene nærmer seg problemstillingen med ulike forskningsspørsmål og gir dermed ulike perspektiver. Arbeidet har et solid teoretisk grunnlag i litteraturstudien, som også sammenfatter tidligere forskning og erfaringer i mer praktisk anvendbare råd for undervisningen. Intervjuundersøkelsen bringer inn lærernes perspektiv. Det er lærerne som vil stå for den faktiske innføringen av programmering i klasserommene, og det er dermed nyttig å forstå hvilke muligheter og utfordringer lærerne ser, og hvordan de forstår begrepet algoritmisk tenkning. Utviklingskomponenten gir arbeidet en forankring til en reell undervisningskontekst med konkrete undervisningsopplegg. Utprøvingen gir dessuten viktige erfaringer fra praksis i fysikk på videregående nivå, noe som foreløpig er mangelvare.

Studien har samtidig en del svakheter og begrensninger. At programmering i skolens fysikkundervisning er et såpass nytt og foreløpig uoversiktlig forskningsområde, har gjort det vanskelig å manøvrere seg blant litteraturen. Som fersk forsker har det vært en bratt læringskurve i å finne gode søkestrategier, og det er sannsynlig at gode arbeider er oversett. Dessuten publiseres det stadig ny forskning som ikke har rukket å bli med i denne oppgaven.

Intervjuundersøkelsen bærer også preg av at lærerne enda ikke hadde rukket å reflektere særlig mye over programmering og algoritmisk tenkning i fysikkfaget. Det har gitt en indikasjon på hvordan lærerne intuitivt forstår begrepene og ser for seg en innføring i skolen, men lærerne ville antakelig kunnet bidratt med flere veloverveide refleksjoner hvis de hadde fått forberede seg mer før intervjuet.

Undervisningsoppleggene som er utviklet, er klart begrenset av hva som har vært praktisk mulig å gjennomføre blant elever som følger dagens læreplaner, og som ikke kan programmering fra før. Arbeidet med denne masteroppgaven har ligget i forkant av de nye læreplanene i fysikk, og oppleggene har derfor tatt utgangspunkt i de temaene som dekkes av dagens læreplaner. Tiden lærerne har kunnet avse til utprøving, sammen med begrensningene i masterarbeidets omfang, medførte at undervisningsoppleggene ble gjennomført med mindre systematiske observasjoner og en lite formell analyse. Studien er kvalitativ, med de fordeler og ulemper det medfører, og det er brukt små bekvemmelighetsutvalg som vi ikke kan vite om er representative. Jeg mener likevel at denne undersøkelsen kan være nyttig og ha overføringsverdi, fordi den gir informasjon om hvordan oppleggene kan utspille seg i klasserommet og hvordan lærerne og elevene opplever undervisningen. Den gir indikasjoner på hvilket nivå programmeringen kan ha uten å overskygge fysikkinnholdet. Erfaringene bør kunne benyttes til senere å gjennomføre mer systematiske datainnsamlinger for blant annet å undersøke elevenes arbeid med programmeringen og deres læringsutbytte.

5.7 Konklusjon

Forskningslitteraturen definerer «algoritmisk tenkning» («computational thinking») som en generell problemløsningsstrategi, som benytter seg av særtrekk ved informasjonsteknologi. Av enkelte anses den som en grunnleggende ferdighet på lik linje med lesing, skriving og regning. En fysikkelev som tenker algoritmisk, kan å avgjøre når det er nyttig å bruke ulike digitale verktøy til datahåndtering, modellering og simulering, problemløsning og systemtenkning. Til dette bruker han også programmering, og han kan overføre disse teknikkene i logisk arbeid og problemløsning også når det ikke benyttes digitale hjelpemidler.

Programmering har blitt et viktig verktøy for fysikere og ingeniører, og har på den måten en naturlig plass også i skolens fysikkfag, slik at elevene får et riktig bilde av hvordan fysikere faktisk arbeider. Med programmering lærer elevene noen generelle teknikker som gjør at de kan løse andre, og mer virkelighetsnære fysikkproblemer, og det åpner for å gjøre større endringer i fysikkfagets form og innhold. Samtidig er det muligheter for at programmering kan bidra til å avdekke enkelte misforståelser, utvide forståelsen for fysiske begreper og fenomener, og å understreke rekkefølgen til årsak og virkning ved kausale forhold. Det er imidlertid begrenset med forskning som undersøker sammenhengen mellom programmering og elevenes læring i fysikk.

Programmering kan innføres som et «krydder» til en ellers tradisjonell tilnærming til faget, eller det kan brukes som et sentralt verktøy og legge premissene for den historien fysikkfaget skal fortelle. Alternativt kan deler av læreplanen, da kanskje spesielt mekanikk, tilnærmes gjennom programmering og numerikk, mens andre deler av læreplanen beholder et kvalitativt preg med programmering som eventuelt krydder.

Det er en fare for at begrepet «algoritmisk tenkning» forstås snevert av fysikklærerne, og at det forbindes mer med algoritmeutvikling enn med den generelle tenkemåten som begrepet er tiltenkt. Det kan være behov for å klargjøre begrepets faktiske innhold, eller det kan vurderes å finne en bedre norsk oversettelse for «computational thinking». Etter inspirasjon fra det svenske begrepet, er «datalogisk tenkning» en mulighet.

Det ser ut til at fysikklærerne i all hovedsak er positive til at elevene skal lære programmering, og de ser også muligheter for hvordan det kan inkluderes i faget – spesielt med tanke på problemløsning i mekanikk og visualisering av fenomener. De er nysgjerrige og har lyst til å prøve det ut, men de kommer selv på få konkrete eksempler, og de frykter at det kan ta mye tid og gjøre fysikkfaget enda vanskeligere.

Undervisningsopplegg med programmering i fysikk kan for eksempel benytte numeriske beregninger i forbindelse med eksperimentelle forsøk, simulering og visualisering, og til å lage et «produkt», som for eksempel en «formelmaskin»/kalkulator. Utviklingen og utprøvingen av undervisningsoppleggene viser at det er mulig å inkludere nokså avansert programmering raskt, samtidig som elevene ser relevansen til fysikk. Mens noen elever viste stor interesse for programmering i fysikk, var det andre elever som opplevde det svært utfordrende. Spesielt det å forstå alle kodene trekkes frem som vanskelig. Det er nødvendig å bruke god tid i oppstarten og tilpasse oppleggene godt for å unngå kognitiv overbelastning. Det er nødvendig å gjøre flere undersøkelser av hvordan oppleggene bidrar til elevenes læring av fysikk.

REFERANSER

Anderson, T. og Shattuck, J. (2012). Design-Based Research: A Decade of Progress in Education Research? *Educational Researcher, 41*(1), 16-25. doi:10.3102/0013189X11428813

Angell, C., Kind, P. M. og Henriksen, E. K. (2008). Implementation of empirical-mathematical modelling in upper secondary physics: Teachers' interpretations and consideration. *NorDiNa, 4*(2), 113-122.

Angell, C., Kind, P. M., Henriksen, E. K. og Guttersrud, Ø. (2008). An empirical-mathematical modelling approach to upper secondary physics. *Physics Education, 43*(3), 256-264.

Balanskat, A., Engelhardt, K. og Ferrari, A. (2017). The integration of Computational Thinking (CT) across school curricula in Europe. *European Schoolnet Perspective*(2).

Barcelos, T. S., Munoz, R., Villarroel, R., Merino, E. og Silveira, I. F. (2018). Mathematics Learning through Computational Thinking Activities: A Systematic Literature Review. *Journal of Universal Computer Science, 24*(7), 815-845.

Barr, V. og Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of computer science education community. *Acm Inroads, 2*(1), 48-54.

Bjørndal, C. R. P. (2013). *Det vurderende øyet – Observasjon, vurdering og utvikling i undervisning og veiledning* (2. utg.). Oslo: Gyldendal Norsk Forlag.

Bocconi, S., Chioccariello, A. og Earp, J. (2018). *The nordic approach to introducing computational thinking and programming in compulsory education*: NORDIC@BETT2018 Steering Group.

Brown, N. C. C. og Wilson, G. (2018). Ten quick tips for teaching programming. *PLOS Computational Biology, 14*(4). doi:10.1371/journal. pcbi.1006023

Bybee, R. W. (2011). Scientific and Engineering Practices in K-12 Classrooms: Understanding «A Framework for K-12 Science Education». *Science Scope, 35*(4), 6-11.

Caballero, M. D., Burk, J. B., Aiken, J. M., Thoms, B. D., Douglas, S. S., Scanlon, E. M. og Schatz, M. F. (2013). Integrating Numerical Computation into the Modeling Instruction Curriculum. *The Physics Teacher, 52*(1), 38-42. doi:10.1119/1.4849153

Caballero, M. D., Kohlmyer, M. A. og Schatz, M. F. (2012). Implementing and assessing computational modeling in introductory mechanics. *Physical Review Special Topics - Physics Education Research, 8*(2), 1-15. doi:10.1103/PhysRevSTPER.8.020106

diSessa, A. A. (2018). Computational Literacy and «The Big Picture» Concerning Computers in Mathematics Education. *Mathematical Thinking and Learning, 20*(1), 3-31. doi:10.1080/10986065.2018.1403544

Dwyer, H., Boe, B., Hill, C., Franklin, D. og Harlow, D. (2013). Computational Thinking for Physics: Programming Models of Physics Phenomenon in Elementary School. I

Engelhardt, Churukian og J. (Eds.) (red.), *2013 PERC Proceedings* (ss. 133-136): American Association of Physics Teachers.

Futschek, G. (2006). *Algorithmic Thinking: The Key for Understanding Computer Science*, Berlin, Heidelberg.

Grover, S. og Pea, R. (2013). Computational Thinking in K–12: A Review of the State of the Field. *Educational Researcher, 42*(1), 38-43. doi:10.3102/0013189X12463051

Hsu, T.-C., Chang, S.-C. og Hung, Y.-T. (2018). How to learn and how to teach computational thinking: Suggestions based on a review of the literature. *Computers and Education, 126*, 296-310. doi:10.1016/j.compedu.2018.07.004

Husén, T. (1988). Research paradigms in education. I J. P. Keeves (red.), *Educational research methodology and measurement. An international handbook* (ss. 17-20). New York: Pergamon Press.

Hutchins, N., Biswas, G., Conlin, L., Emara, M., Grover, S., Basu, S. og McElhaney, K. (2018). Studying Synergistic Learning of Physics and Computational Thinking in a Learning by Modeling Environment. I J. C. e. a. Yang (red.), *Proceedings of the 26th International Conference on Computers in Education* (ss. 153-162). Philippines: Asia-Pacific Society for Computers in Education.

Jenkins, T. (2002). On the difficulty of learning to program. *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, 53-58.

Kelleher, C. og Pausch, R. (2005). Lowering the Barriers to Programming: a survey of programming environments and languages for novice programmers. *ACM Computing Surveys, 37*(2), 83-137. doi:10.1145/1089733.1089734

Kjällander, S., Åkerfeldt, A. og Petersen, P. (2016, 23. mars). Översikt avseende forskning och erfarenheter kring programmering i förskola och grundskola. Hentet 28. oktober 2018 fra https://natverk.dfs.se/system/files/oversikt_programmering_i_skolan.pdf

Kortemeyer, G. og Kortemeyer, A. F. (2018). The nature of collaborations on programming assignments in introductory physics courses: a case study. *European Journal of Physics, 39*(5). doi:10.1088/1361-6404/aad511

Kreyzig, E. (2011). *Advanced Engineering Mathematics* (10. utg.). Hoboken: John Wiley & sons, inc.

Kuhn, T. S. (1970). *The Structure of Scientific Revolutions* (O. Neurath red. 2. utg. Vol. 2). USA: The University of Chicago Press.

Kunnskapsdepartementet. (2017, 25. august). Framtid, fornyelse og digitalisering. Digitaliseringsstrategi for grunnopplæringen 2017-2021. Hentet 14. oktober 2018 fra https://www.regjeringen.no/contentassets/dc02a65c18a7464db394766247e5f5fc/kd_framtid_fornyelse_digitalisering_nett.pdf

Kunnskapsdepartementet. (2018, 26. juni 2018). Fornyer innholdet i skolen. Hentet 26. september 2018 fra https://www.regjeringen.no/no/aktuelt/fornyer-innholdet-i-skolen/id2606028/

KURT - Kompetansesenter for undervisning i realfag og teknologi. (2019, 24. januar). ProFag - realfaglig programmering. Hentet 30. januar 2019 fra https://www.mn.uio.no/om/samarbeid/tilbud-skoler/kurt/kompetanseheving-skole/profag/index.html

Kvale, S. og Brinkmann, S. (2015). *Det kvalitative forskningsintervju* (T. M. Anderssen og J. Rygge, overs. 3. utg.). Oslo: Gyldendal akademisk.

Lahtinen, E., Ala-Mutka, K. og Järvinen, H.-M. (2005). A study of the difficulties of novice programmers. *Acm Sigcse Bulletin, 37*(3), 14-18.

Langtangen, H. P. (2009). *A primer on scientific programming with python*. Berlin: Springer.

Lindstrøm, T. (2006). *Kalkulus* (3. utg.). Oslo: Universitetsforlaget.

Malthe-Sørenssen, A. (2015). *Elementary mechanics using Python*. Cham: Springer.

Malthe-Sørenssen, A., Hjorth-Jensen, M., Langtangen, H. P. og Mørken, K. (2015). Integrasjon av beregninger i fysikkundervisningen. *Uniped, 38*(04), 303-310.

National Research Council (NRC). (2012). *A framework for K-12 science education: Practices, crosscutting concepts, and core ideas.* Washington, D.C.: The National Academies Press.

NESH – Den nasjonale forskningsetiske komité for samfunnsvitenskap og humaniora. (2016, april). Forskningsetiske retningslinjer for samfunnsvitenskap, humaniora, juss og teologi. Hentet 18. november 2018 fra https://www.etikkom.no/globalassets/documents/publikasjoner-som-pdf/60125_fek_retningslinjer_nesh_digital.pdf

Nilsson, J. A. (2018, 15. september). Programmering som moment i gymnasiematematiken – En intervjustudie med berörda lärare. Hentet 19. november 2018 fra http://hh.diva-portal.org/smash/get/diva2:1248488/FULLTEXT02.pdf

NOU 2015: 8. (2015). *Fremtidens skole – Fornyelse av fag og kompetanser*. Oslo: Kunnskapsdepartementet Hentet 11. januar 2019 fra https://www.regjeringen.no/contentassets/da148fec8c4a4ab88daa8b677a700292/no/pdfs/nou201520150008000dddpdfs.pdf.

NTNU. (2018, 25. september). Retningslinje for behandling av personopplysninger. Hentet 18. november 2018 fra https://innsida.ntnu.no/wiki/-/wiki/Norsk/Retningslinje+for+behandling+av+personopplysninger

Orban, C. M., Teeling-Smith, R. M., Smith, J. R. H. og Porter, C. D. (2018). A hybrid approach for using programming exercises in introductory physics. *American Journal of Physics, 86*(11), 831-838. doi:10.1119/1.5058449

Papert, S. (1980). *Mindstorms. Children, Computers, and Powerful Ideas*. New York: Basic Books, Inc., Publishers.

Pierson, A. E. og Clark, D. B. (2018). Engaging students in computational modeling: The role of an external audience in shaping conceptual learning, model quality, and classroom discourse. *Science Education, 102*(6), 1336-1362. doi:10.1002/sce.21476

Piteira, M. og Costa, C. (2013). Learning computer programming: study of difficulties in learning programming. *Proceedings of the 2013 International Conference on Information Systems and Design of Communication*, 75-80.

Postholm, M. B. (2005). Analyse i kvalitativ forskning *Kvalitativ metode – en innføring med fokus på fenomenologi, etnografi og kasusstudier* (ss. 86-106): Universitetsforlaget.

Quarteroni, A., Sacco, R. og Saleri, F. (2007). *Numerical Mathematics*. New York: Springer Berlin Heidelberg.

Robson, C. og McCartan, K. (2016). *Real world research – A Resource for Users of Soscial Research Methods in Applied Settings* (4. utg.). Chichester, England: John Wiley & Sons Ltd.

Rossen, E. (2017, 18. oktober). Programmering - it. I Store norske leksikon. Hentet 28. oktober 2018 fra https://snl.no/programmering_-_IT

Roundy, D., Krebs, E. J., Schulte, J. B. og Mulder, G. S. (2015, 21.-24. okt.). *Look ma, no templates! Problem-based learning of computational physics for novice programmers.* Paper presented at the 2015 IEEE Frontiers in Education Conference (FIE).

Sadiku, M. N. O., Shadare, A. E. og Musa, S. M. (2017). Computational Physics : An Introduction. *International Journal of Engineering Research, 6*(9), 427-428. doi:10.5958/2319-6890.2017.00054.X

Sanne, A., Berge, O., Bungum, B., Jørgensen, E. C., Kluge, A., Kristensen, T. E., . . . Voll, L. O. (2016). *Teknologi og programmering for alle. En faggjennomgang med forslag til endringer i grunnopplæringen.* Hentet 13. oktober 2018 fra https://www.udir.no/tall-og-forskning/finn-forskning/rapporter/teknologi-og-programmering-for-alle

Savitch, W. (2013). *Absolute C++* (5. utg.). Essex, England: Pearson Education Limited.

Sengupta, P., Kinnebrew, J. S., Basu, S., Biswas, G. og Clark, D. (2013). Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies, 18*(2), 351-380. doi:10.1007/s10639-012-9240-x

Sherin, B. L. (2001). A comparison of programming languages and algebraic notation as expressive languages for physics. *International Journal of Computers for Mathematical Learning, 6*(1), 1-61.

Stolpe, K. (2018, 1. februar). Programmering – elever blir bra på det de övar! Hentet 6. november 2018 fra https://liu.se/cetis/nyhetsbrev/2018-1-programmering.shtml

Store norske leksikon. (2018, 20. februar). Algoritme. Hentet 28. oktober 2018 fra https://snl.no/algoritme

Sørby, S. A. (2010). *Beregningsorientert fysikk i bachelorkurs ved Universitetet i Oslo.* (Mastergrad), Universitetet i Oslo.

Sørby, S. A. og Angell, C. (2012). Undergraduate student's challenges with computational modelling in physics. *NorDiNa, 8*(3), 283-296.

Taub, R., Armoni, M., Bagno, E. og Ben-Ari, M. M. (2015). The effect of computer science on physics learning in a computational science environment. *Computers & Education, 87*, 10-23.

Tellefsen, C. W. (2018, 24. oktober). Realfaglig programmering. *Universitetet i Oslo. FIKS - Forskning, innovasjon og kompetanseutvikling i skolen.* Hentet 23. januar 2019 fra https://www.uv.uio.no/forskning/satsinger/fiks/kunnskapsbase/realfaglig-programmering/

Utdanningsdirektoratet. (2017a, 18. oktober). Fagfornyelsen – Matematikk. Hentet 27. september 2018 fra http://udirbloggen.no/matematikk

Utdanningsdirektoratet. (2017b, 17. oktober). Fagfornyelsen – Naturfag. Hentet 27. september 2018 fra http://udirbloggen.no/naturfag

Utdanningsdirektoratet. (2017c, 21. september). Kjerneelementer i matematikk, men hvorfor programmering? Hentet 14. oktober 2018 fra http://udirbloggen.no/kjerneelementer-i-matematikk-men-hvorfor-programmering/

Utdanningsdirektoratet. (2018a, 18. oktober). Fagfornyelsen – innspillsrunde skisser til læreplaner i naturfag. Hentet 31. oktober 2018 fra https://hoering.udir.no/Hoering/v2/277?notatId=531

Utdanningsdirektoratet. (2018b, 18. oktober). Fagfornyinga – innspelsrunde skisser til læreplanar i matematikk. Hentet 31. oktober 2018 fra https://hoering.udir.no/Hoering/v2/286?notatId=573

Utdanningsdirektoratet. (2019a, 27. mars). Algoritmisk tenkning. Hentet 25. april 2019 fra https://www.udir.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-kompetanse/algoritmisk-tenkning/

Utdanningsdirektoratet. (2019b, 18. mars). Høring – Læreplan i naturfag. Hentet 18. mars 2019 fra https://hoering.udir.no/Hoering/v2/346?notatId=683

Utdanningsdirektoratet. (2019c, 18. mars). Høring – Læreplaner i matematikk. Hentet 18. mars 2019 fra https://hoering.udir.no/Hoering/v2/343?notatId=656

Wagner, S. P. (1998). Robotics and children: science achievement and problem solving. *Journal of Computing in Childhood Education, 9*(2), 149-192.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L. og Wilensky, U. (2016). Defining Computational Thinking for Mathematics and Science Classrooms. *Journal of Science Education and Technology, 25*(1), 127-147. doi:10.1007/s10956-015-9581-5

Wilensky, U., Brady, C. E. og Horn, M. S. (2014). Fostering computational literacy in science classrooms. *Communications of the ACM, 57*(8), 24-28.

Wing, J. M. (2006). Computational Thinking. *Communications of the ACM, 49*(3), 33-35.

Wing, J. M. (2014, 10. januar). Computational thinking benefits society. *Social issues in computing. 40th anniversary blog.* Hentet 14. januar 2019 fra http://socialissues.cs.toronto.edu/index.html%3Fp=279.html

Yaşar, O. (2013). Teaching science through computation. *International Journal of Science, Technology and Society, 1*(1), 9-18. doi:10.11648/j.ijsts.20130101.12

Zaharija, G., Mladenović, S. og Boljat, I. (2013). Introducing basic Programming Concepts to Elementary School Children. *Procedia - Social and Behavioral Sciences, 106*, 1576-1584. doi:https://doi.org/10.1016/j.sbspro.2013.12.178

Aalmen, F. (2010). *Bruk av drama for undervisning av naturvitenskapens egenart.* (Mastergrad), Norges teknisk-naturvitenskapelige universitet.

VEDLEGG

VEDLEGG A: Intervjuguide

Takk for at du vil delta i dette intervjuet om programmering og «algoritmisk tenkning» i fysikkfaget i skolen.

Jeg vil informere om at du forblir anonym og at jeg har taushetsplikt. Kun anonymiserte utdrag av transkripsjon av intervjuet kan offentliggjøres i en rapport.

Underskrive samtykkeerklæring til lydopptak av intervjuet.

1. Bakgrunn og presentasjon av kjerneelementer:

   Høsten 2017 lanserte regjeringen en strategi for digitalisering av norsk skole. Strategien peker på at elevene skal få økt opplæring i teknologi, og herunder programmering og det som kalles algoritmisk tenkning, for å svare på samfunnets raske teknologiske utvikling, integrert i allerede eksisterende fag (matematikk, naturfag, k&h). Fysikk er foreløpig ikke med i fagfornyelsen, men vi antar at algoritmisk tenkning også vil inkluderes i programfagene i neste omgang.

2. Har du selv noen erfaring med programmering?
3. Et begrep innenfor området er «algoritmisk tenkning». Vet du hva det er?
4. Hva tenker du om innføring av programmering og algoritmisk tenkning i fysikk?
   a. Hvilke muligheter og fordeler kan det gi?
   b. Kan det bidra til økt læring i fysikk?
   c. Er det noen tema det egner seg å undervise med programmering?
   d. Kan det bidra til allmenndannelse?
   e. Hvilke utfordringer kan du se for deg i undervisningssammenheng?

5. Kan du tenke deg noen sentrale prinsipper/kjøreregler for hvordan undervise programmering i fysikk?
   a. Hvis du skulle planlegge et undervisningsopplegg med algoritmisk tenkemåte for å dekke kompetansemål i dagens læreplan. Hvordan ville du gått frem?
   b. Hvordan vil du vurdere elevene i programmering og algoritmisk tenkning?

6. Nå skal du få opplest fire kategorier i en modell for hva algoritmisk tenkning kan være i matematikk og naturfag. Kan du forsøke å beskrive hva du opplever dette kan innebære i undervisning av algoritmisk tenkning i fysikk?
   a. Datahåndtering
   b. Modellering og simulering
   c. Digital problemløsning
   d. Systemtenkning

Takk for at du tok deg tid. Til slutt vil jeg spørre om det er noen aspekter ved programmering i fysikk i norsk skole du har synspunkter om, som du ikke føler er dekket så langt i intervjuet.

VEDLEGG B: Samtykkeerklæring

Noen elementer er utelatt for å ivareta anonymitet.

Vil du delta i forskningsprosjektet «Programmering i fysikkundervisning»?

Til deg som er fysikklærer i videregående skole

Jeg er student på lektorprogrammet i realfag ved NTNU og skal gjennomføre et kort forskningsprosjekt om programmering i norsk fysikkundervisning. I dette skrivet gir jeg deg informasjon om målene for prosjektet og hva deltakelse vil innebære for deg.

Formål
Høsten 2017 presenterte regjeringen en strategi for å innføre mer koding og teknologi i norsk skole. I forbindelse med arbeidet om nye læreplaner, ble det våren 2018 slått fast av Kunnskapsdepartementet at «programmering og algoritmisk tankegang» skal inngå som en del av kjerneelementene i fellesfagene for matematikk, naturfag og kunst og håndverk. Vi forventer at programmering også skal innføres i programfag for fysikk ved en fremtidig fornyelse av programfagene i norsk videregående skole.

I denne forbindelse skal jeg våren 2018 utføre et mastergradsarbeid om programmering og algoritmisk tenkning i fysikk. Jeg er derfor interessert i å undersøke hva fysikklærere tenker om temaet og hvilke muligheter og utfordringer de ser for programmering og læring av fysikk.

Resultatene av studien vil bli brukt i en eksamensbesvarelse i et kurs om forskningsmetoder ved NTNU. Resultatene kan, senere, bli brukt i en masteroppgave.

Hvem er ansvarlig for forskningsprosjektet?
NTNU, Institutt for lærerutdanning ([fjernet]) er ansvarlig for prosjektet.

Hvorfor får du spørsmål om å delta?
Du er ansatt som lærer med undervisningskompetanse i fysikk og [fjernet].

Hva innebærer det for deg å delta?
Hvis du velger å delta i prosjektet, innebærer det at du deltar i et intervju med meg, der du får noen spørsmål om hvordan du tenker programmering kan være en del av fremtidens fysikkfag. Det er ingen krav til at du selv har programmeringserfaring – det er dine meninger som er viktig. Det vil ta deg ca. 30 minutter. Jeg tar lydopptak og notater fra intervjuet.

Det er frivillig å delta
Det er frivillig å delta i prosjektet. Hvis du velger å delta, kan du når som helst trekke samtykket tilbake uten å oppgi noen grunn. Alle opplysninger om deg vil da bli anonymisert. Det vil ikke ha noen negative konsekvenser for deg hvis du ikke vil delta eller senere velger å trekke deg.

Ditt personvern – hvordan vi oppbevarer og bruker dine opplysninger
Vi vil bare bruke opplysningene vi samler inn til formålene vi har fortalt om i dette skrivet. Vi behandler opplysningene konfidensielt og i samsvar med personvernregelverket.

Det er kun jeg og mine to veiledere ved Institutt for lærerutdanning, NTNU, som vil ha tilgang til innsamlet datamateriale. Lydopptakene vil oppbevares forsvarlig på en ekstern, kryptert og innelåst lagringsenhet og snarest transkribert i anonymisert form. Deretter blir lydopptakene slettet og lagringsenheten formatert. Du som deltaker vil ikke kunne gjenkjennes i en eventuell publikasjon.

Hva skjer med opplysningene dine når vi avslutter forskningsprosjektet?
Prosjektet skal etter planen avsluttes 1. januar 2019. Da vil de originale lydopptakene være slettet, og kun anonymiserte transkripsjoner vil være tilgjengelige.

Dine rettigheter
Så lenge du kan identifiseres i datamaterialet, har du rett til:

- innsyn i hvilke personopplysninger som er registrert om deg,
- å få rettet personopplysninger om deg,
- få slettet personopplysninger om deg,
- få utlevert en kopi av dine personopplysninger (dataportabilitet), og
- å sende klage til personvernombudet eller Datatilsynet om behandlingen av dine personopplysninger.

Hva gir oss rett til å behandle personopplysninger om deg?
Vi behandler opplysninger om deg basert på ditt samtykke.

På oppdrag fra Institutt for lærerutdanning har NSD – Norsk senter for forskningsdata AS vurdert at behandlingen av personopplysninger i dette prosjektet er i samsvar med personvernregelverket.

Hvor kan jeg finne ut mer?
Hvis du har spørsmål til studien, eller ønsker å benytte deg av dine rettigheter, ta kontakt med:

- Institutt for lærerutdanning ved [fjernet]
- Vårt personvernombud: [fjernet]
- NSD – Norsk senter for forskningsdata AS, på e-post (personverntjenester@nsd.no) eller telefon: 55 58 21 17.

Med vennlig hilsen

Prosjektansvarlig	Student
[fjernet]	[fjernet]

Samtykkeerklæring

Jeg har mottatt og forstått informasjon om prosjektet «Programmering i fysikkundervisning», og har fått anledning til å stille spørsmål. Jeg samtykker til:

- å delta i et intervju og at et lydopptak av intervjuet behandles frem til prosjektet er avsluttet, 1. januar 2019

(Signert av prosjektdeltaker, dato)

VEDLEGG C: Spørreundersøkelse

I dag har du og klassen din arbeidet med programmering i fysikkundervisningen. Til slutt vil vi stille noen spørsmål om hvordan du opplevde disse timene og hva du lærte. Svarene er anonyme og kan bli publisert i en mastergrad ved NTNU. På forhånd takk!

---

1. Hvilken karakter regner du med å få i fysikk standpunktkarakter i år?

   ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6

2. Hvor mye erfaring hadde du med programmering før i dag?

○ 1	○ 2	○ 3	○ 4	○ 5	○ 6
Ingen erfaring					Svært mye erfaring

   Kommentarer?

3. Hvor utfordrende opplevde du at undervisningen med programmering var?

○ 1	○ 2	○ 3	○ 4	○ 5	○ 6
Svært enkel					Svært utfordrende

4. Hva var mest utfordrende?

5. Hva synes du var mest interessant?

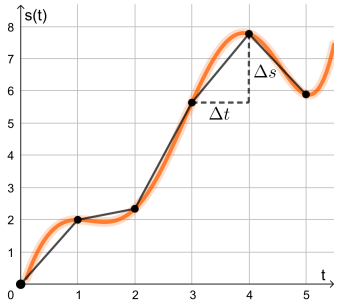
6. Hva synes du opplegget handlet mest om i dag?

Fysikk	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6
	Svært liten grad					Svært stor grad
Programmering	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6
	Svært liten grad					Svært stor grad
Matematikk	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6
	Svært liten grad					Svært stor grad

7. Nevn noen bruksområder du tror programmering kan ha i fysikk.
   Lærte du noe nytt om dette i dag?

VEDLEGG D: Oppgaveark for opplegg 1

NUMERISK DERIVASJON



ANALYTISK DERIVASJON

$$v(t) = \lim_{\Delta t \to 0} \frac{s(t+\Delta t) - s(t)}{\Delta t}$$

NUMERISK DERIVASJON

$$v(t_i) \approx \frac{s(t_{i+1}) - s(t_i)}{t_{i+1} - t_i}$$

1. Utvid programkoden om Warholms 400-meter hekk som du skrev sammen med læreren slik at du får frem en graf over Warholms akselerasjon.

Hva forteller grafen om Warholms løp?

Hva kjennetegner et datamateriale som egner seg for numerisk derivasjon? Hva kan du si om datamaterialet du har fått utdelt?

2. Finn utstyr på fysikk-laben og undersøk en rettlinjet bevegelse. Forslag:

- Et lodd som svinger i en fjær
- En gjenstand som faller
- En kloss som glir ned et skråplan
- En bil som triller
- Et vertikalt hopp
- En elev som løper frem og tilbake

Film bevegelsen med et kamera/smarttelefon og bruk programmet Tracker til å spore bevegelsen. Sørg for godt lys og at bevegelsen skjer i det planet hvor kameraet filmer. Eksporter dataene til en `txt`-fil som du laster inn i Python.

Analyser datamaterialet ved hjelp av numerisk derivasjon.
Kan du bruke datamaterialet til å si noe om kreftene som virker og energien i bevegelsen?

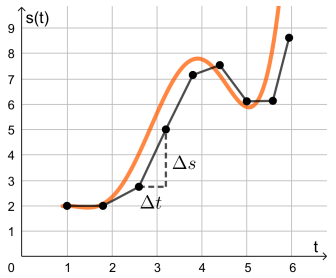
3. En lærer som ønsker å komme i form, har sporet løpeturen sin med en GPS-app.
I filen `løpetur.txt` finner du tiden etter start i sekunder og farten ved det gitte tidspunktet, målt i antall minutter og sekunder per kilometer.

Importer datamaterialet med Python-kode.
Lag en algoritme i programkode for å beregne farten i meter per sekund.

Kan du finne ut hvor langt læreren løp?
Du kan ta utgangspunkt i uttrykket for numerisk derivasjon og lage en algoritme for å beregne strekningen læreren har løpt ved hvert tidspunkt numerisk.

VEDLEGG E: Oppgaveark for opplegg 2

BEVEGELSE I INHOMOGENT FELT



NUMERISK INTEGRASJON
Euler-metoden

ny = gammel + endring

$$s(t_{i+1}) = s(t_i) + v(t_i) \cdot \Delta t$$

1. Skriv ferdig funksjonen `distance(A, B)` slik at den returnerer avstanden mellom de to punktene `A` og `B`. Du henter ut x-koordinaten ved å bruke `A[0]` og y-koordinaten ved å bruke `A[1]`. Det kan være lurt å tegne en figur.

2. Skriv ferdig funksjonen `coulombforce(q, A, Q, B)` slik at den regner ut den elektriske kraften som virker på ladning q i punkt A fra ladning Q i punkt B.
   Det kan være lurt å tegne en figur.
   Funksjonen skal returnere x- og y-komponenten som en vektor (array): `[Fx, Fy]`.

Du har fått koden som skal brukes til å simulere bevegelsen til en ladning i et elektrisk felt i filen ladning.py. Noe av koden er ferdig, mens noe må du fullføre.

3. Legg inn funksjonene `distance()` og `coulombforce()` i `ladning.py`.
   Fyll ut konstanter og startverdier, slik at q er et positron (antielektron) i posisjon $A(2, 2)$ og Q er et positron i posisjon $B(0, 0)$

4. Animasjonen kjøres ved hjelp av en `while`-løkke, som regner ut ny kraft, akselerasjon, fart og posisjon etter `dt = 0.01` sekunder. Partikkelen flyttes til nye koordinater med funksjonen `place()`

   Bruk funksjonen `coulombforce()` til å regne ut kraften `F` på q inne i while-løkken. Regn også ut akselerasjonen `a`, farten `v` og posisjonen `s` ved å bruke Euler-metoden inne i while-løkken.

5. Kjør simuleringen.
   Velg ulike verdier for startposisjonen A, utgangsfarten v og ladningene q og Q.
   Sjekk at simuleringen oppfører seg som forventet, og rett eventuelle feil.
   Hva skjer når partiklene kolliderer. Hvordan kan det forklares?

6. Velg en startposisjon og startfart slik at den ene partikkelen går i en sirkelbane rundt den andre partikkelen. Kan du klare å lage en ellipsebane?

7. Utvid koden med en (eller flere) partikler i ro. Forsøk å få den bevegelige partikkelen til å svinge frem og tilbake. Kan du få partikkelen til å gå i en åttetallsbevegelse?

UTFORDRING: En ladning som akselererer, avgir egentlig elektromagnetisk stråling og vil dermed miste noe kinetisk energi. Søk etter en formel for hvor mye energi som går tapt, og forbedre simuleringen slik at den tar høyde for strålingstapet.

VEDLEGG F: Oppgaveark for opplegg 3

FOTONOMATOR

FARGE	NAVN	BØLGELENGDER (nm)	RGB-KODE
	Usynlig	> 700	(0, 0, 0)
	Rød	630 – 700	(1, 0, 0)
	Oransje	590 – 630	(1, 0.5, 0)
	Gul	560 – 590	(1, 1, 0)
	Grønn	520 – 560	(0, 1, 0)
	Turkis	490 – 520	(0, 1, 1)
	Blå	450 – 490	(0, 0, 1)
	Fiolett	400 – 450	(1, 0, 1)
	Usynlig	< 400	(0, 0, 0)

1. Lag et program som ber brukeren taste inn en bølgelengde i nanometer ved å bruke koden `input( )`.
   Konverter brukerinntastingen fra tekst (*string*) til desimaltall (*float*) ved å bruke koden `float( )`. Lagre bølgelengden i en variabel.

2. Bruk bølgelengden til å beregne frekvensen og fotonenergien.
   Skriv ut bølgelengde, frekvens og energi på en pen måte i konsollen.

3. Lag en kode som avgjør fargen på fotonet og lagrer fargenavnet i en tekst-variabel.
   Skriv ut fargenavnet sammen med bølgelengde, frekvens og energi i konsollen.

4. Utvid programmet slik at det definerer tre variabler, R, G og B, med de riktige RGB-verdiene til fargene. Bruk koden nedenfor til å vise RGB-fargen i konsollen.

```
1. fig = figure()
2. ax = fig.add_subplot(1, 1, 1)
3. ax.set_facecolor( (R,G,B) )
4. show()
```

5. Utvid programmet slik at det viser glidende fargeoverganger.
   Du må lage en algoritme slik at RGB-koden gradvis endres mellom fargene.

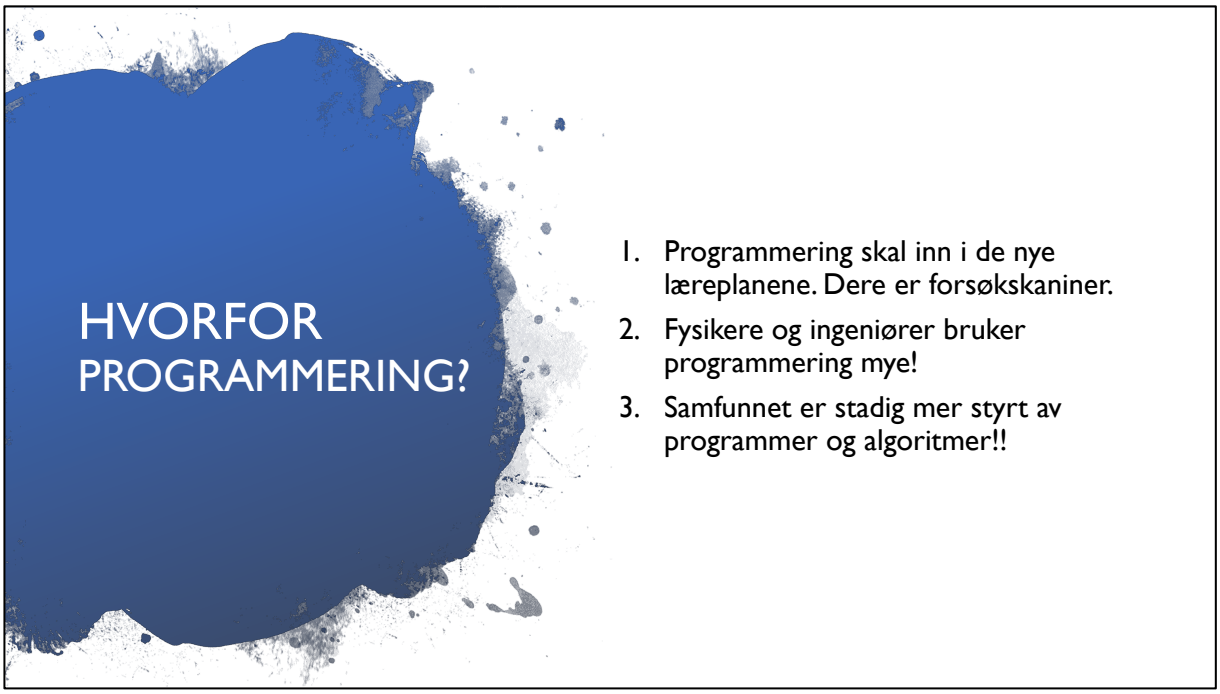
   RGB står for Rød, Grønn, Blå, hvor hver av verdiene går fra 0 til 1. Når R-verdien er 1, betyr det maksimal rødfarge. Hvis R-verdien er 0.6, betyr det 60 % rødfarge.
   Når for eksempel bølgelengden øker fra 630 nm til 560 nm, skal RGB-koden gå gradvis fra helt rød, (1, 0, 0), til helt gul, som er en blanding av helt rød og helt grønn (1, 1, 0).

   Utvid programmet slik at det fortsetter å spørre om nye bølgelengder helt til brukeren skriver inn 0 for å avslutte programmet.

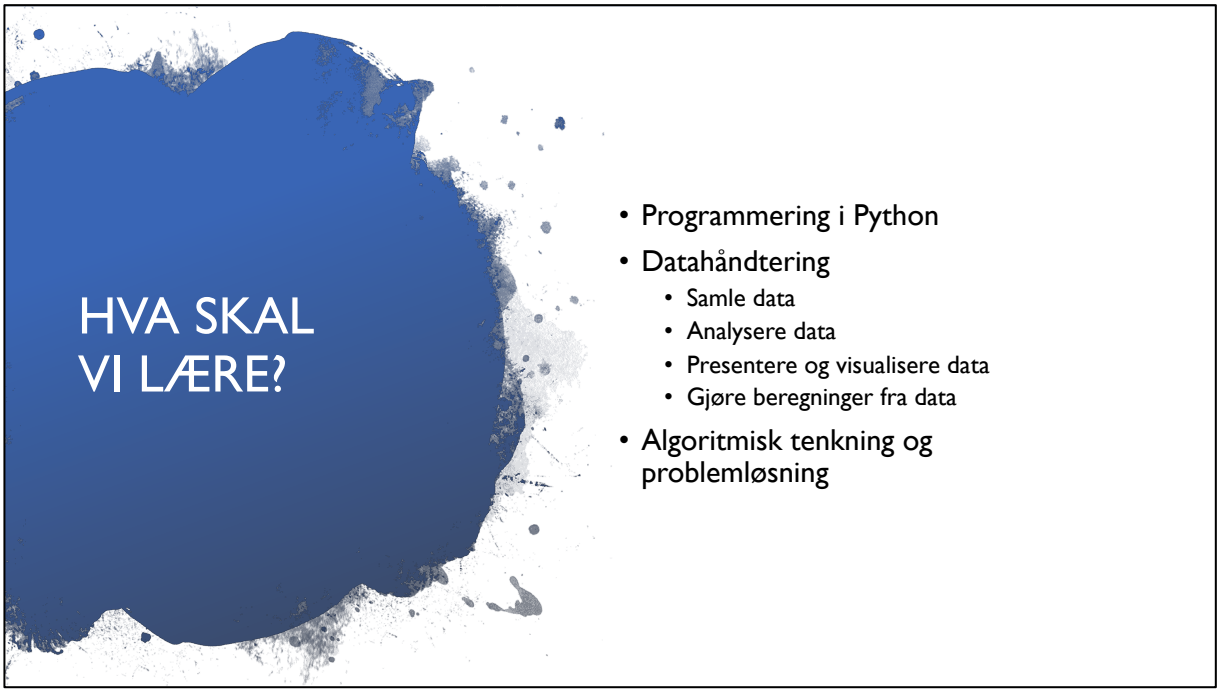
VEDLEGG G: Lysbildepresentasjon for opplegg 1

Lysbilde G-1

Lysbilde G-2

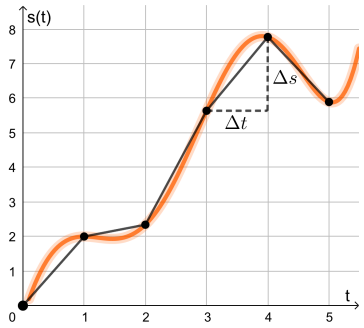


Lysbilde G-3



Lysbilde G-4

DERIVASJON



- ANALYTISK DERIVASJON

$$v(t) = \lim_{\Delta t \to 0} \frac{s(t+\Delta t) - s(t)}{\Delta t}$$

- NUMERISK DERIVASJON

$$v(t_i) \approx \frac{s(t_{i+1}) - s(t_i)}{t_{i+1} - t_i}$$

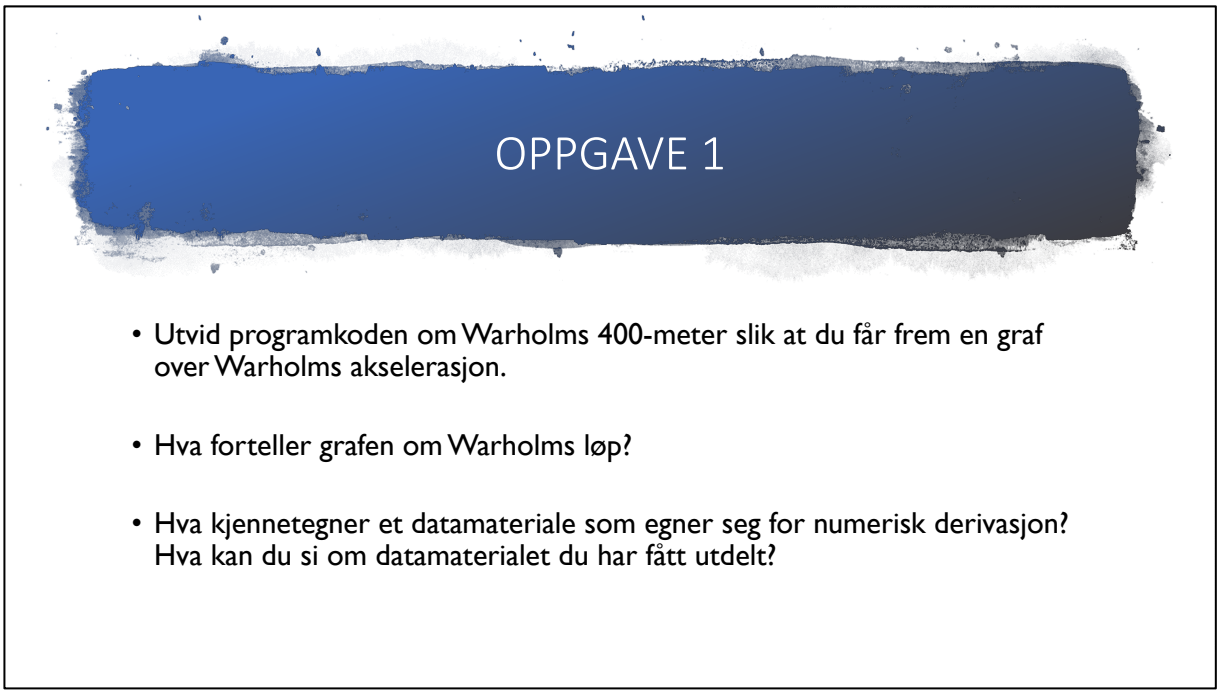
Lysbilde G-5

WARHOLMS 400-METER

- LAST NED *Warholm400.txt* FRA
  http://tinyurl.com/progfys1

```
from pylab import *

# Laste inn data fra Warholms løp
data = loadtxt("Warholm400.txt", delimiter=" ")
posisjon = data[:,0]
tid = data[:,1]
n = len(tid)

#Tegne grafen til tid/posisjon
plot(tid, posisjon, 'ro-')
xlabel('Tid i sekunder')
ylabel('Strekning i meter')
show()

#Beregne fart
fart = zeros(n-1)
for i in range(n-1):
    fart[i] = (posisjon[i+1]-posisjon[i])/(tid[i+1]-tid[i])

#Tegne grafen til tid/fart
plot(tid[0:-1], fart, 'bo-')
xlabel('Tid i sekunder')
ylabel('Fart i m/s')
show()
```

Lysbilde G-6

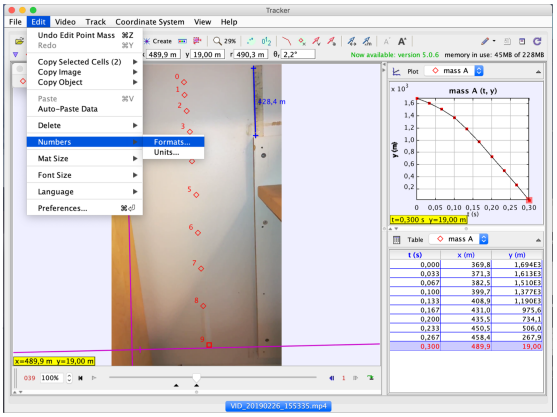


Lysbilde G-7

```
from pylab import *

# Laste inn data fra Warholms løp
data = loadtxt("Warholm400.txt", delimiter=" ")
posisjon = data[:,0]
tid = data[:,1]
n = len(tid)

#Tegne grafen til tid/posisjon
plot(tid, posisjon, 'ro-')
show()

#Beregne fart
fart = zeros(n-1)
for i in range(n-1):
    fart[i] = (posisjon[i+1]-posisjon[i])/(tid[i+1]-tid[i])

#Tegne grafen til tid/fart
plot(tid[0:-1], fart, 'bo-')
show()

#Beregne akselerasjon
akselerasjon = zeros(n-2)
for i in range(n-2):
    akselerasjon[i] = (fart[i+1]-fart[i])/(tid[i+1]-tid[i])

#Tegne grafen til tid/akselerasjon
plot(tid[0:-2], akselerasjon, 'go-')
show()
```

Lysbilde G-8

TRACKER



Lysbilde G-9

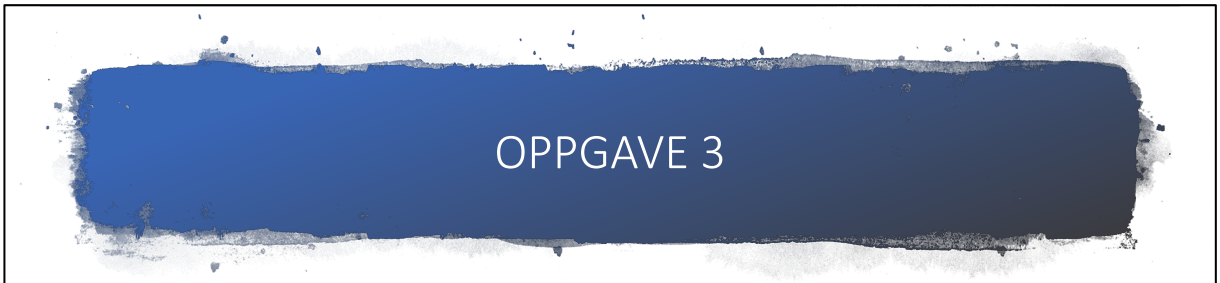
OPPGAVE 2

- Finn utstyr på fysikk-laben og undersøk en rettlinjet bevegelse. Forslag:
  - Et lodd som svinger i en fjær
  - En gjenstand som faller
  - En kloss som glir ned et skråplan
  - En bil som triller
  - Et vertikalt hopp
  - En elev som løper frem og tilbake
- Film bevegelsen og spor den med Tracker.
- Sett desimaltegn til punktum i Tracker. Eksporter dataene til en txt-fil som du laster inn i Python.
- Analyser datamaterialet ved hjelp av numerisk derivasjon.
  Kan du bruke datamaterialet til å si noe om kreftene som virker og energien i bevegelsen?

Lysbilde G-10

```python
from pylab import *


data = loadtxt('fallendehanske.txt', skiprows=2)
t = data[:,0]
x = data[:,1]
y = data[:,2]


plot(t,y,'ro-')
show()
```

Lysbilde G-11



- En lærer som ønsker å komme i form, har sporet løpeturen sin med en GPS-app. I filen `løpetur.txt` finner du tiden etter start i sekunder og farten ved det gitte tidspunktet, målt i antall minutter og sekunder per kilometer.
- Lag en algoritme i programkode for å beregne farten i meter per sekund.
- Kan du finne ut hvor langt læreren løp?

Lysbilde G-12

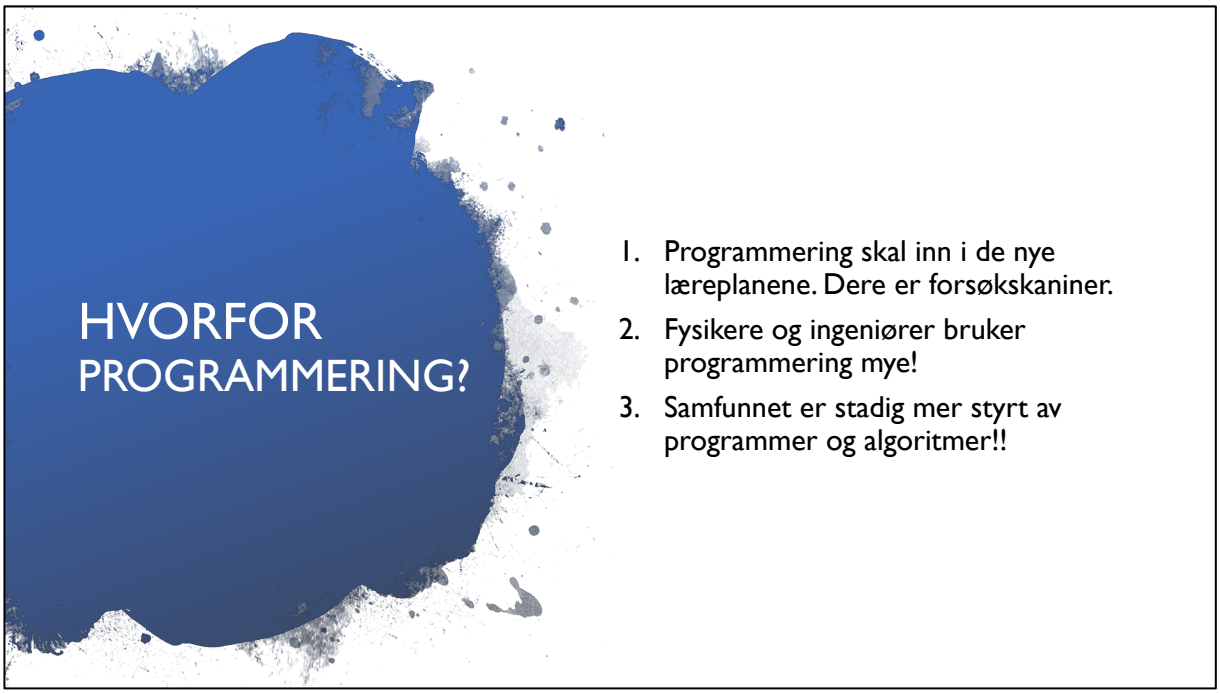
```python
from pylab import *

#Henter data fra lærerens løpetur og antall datapunkter
data = loadtxt("løpetur.txt", delimiter=" ")
n = len(data)
dt = 10

#Deler opp data, kolonne 0 er tid, kolonne 1 er minutter pr km, kolonne 2 er sekunder pr km
tid = data[:,0]
minutter = data[:,1]
sekunder = data[:,2]

#Regner fra tid pr km til fart i m/s
fart = 1000/(minutter*60+sekunder)

#Tegne grafen over fart og tid (i minutter)
plot(tid/60,fart)
xlabel('Tid i minutter')
ylabel('Fart i m/s')
show()

#Lage tomme lister for strekning og akselerasjon
s = zeros(n)
a = zeros(n)

#Beregne strekning med ny = gammel + endring
#Beregne akselersajon med derivert = endring / tid
for i in range(len(tid)-1):
    s[i+1] = s[i] + fart[i] * dt
    a[i+1] = (fart[i+1]-fart[i])/dt

#Tegne grafen til strekning i km
plot(tid/60,s/1000)
xlabel('Tid i minutter')
ylabel('Strekning i km')
show()

#Tegne grafen til akselerasjon i m/s^2
plot(tid/60,a)
xlabel('Tid i minutter')
ylabel('Akselerasjon i $m/s^2$')
show()

print('Læreren løp ' + str(round(s[-1]/1000,1)) + ' km til sammen')
```

Lysbilde G-13

VEDLEGG H: Lysbildepresentasjon for opplegg 2

Lysbilde H-1

Lysbilde H-2



Lysbilde H-3

Lysbilde H-4

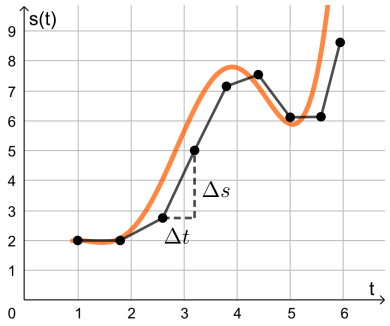
FUNKSJONER I PYTHON

- Likner på funksjoner i matematikk
- Sender inn én eller flere verdier
- Får ut en annen verdi

```python
from pylab import *

## Funksjon
def akselerasjon(t):
    return t**2 - 1

## Teste funksjonen
print('Akselerasjonen etter 2 sekunder er ', akselerasjon(2))
```

Lysbilde H-5

NUMERISK INTEGRASJON (Euler-metoden)



ny = gammel + endring

$$s(t_{i+1}) = s(t_i) + v(t_i) \cdot \Delta t$$

$$v(t_{i+1}) = v(t_i) + a(t_i) \cdot \Delta t$$

Lysbilde H-6

```python
from pylab import *

## Funksjon
def akselerasjon(t):
    return t**2 - 1

## Teste funksjonen
print('Akselerasjonen etter 2 sekunder er ', akselerasjon(2))

## Eulermetoden
## Regne ut posisjon numerisk (når akselerasjon ikke er konstant)
v = 0
s = 0
t = 0

dt = 0.1

while t <= 10:
    t = t + dt
    v = v + akselerasjon(t)*dt
    s = s + v*dt

print('Posisjonen etter 10 sekunder er ', s)
```

Lysbilde H-7

- Lag en funksjon `distance(A, B)` som regner ut avstanden mellom to punkter `A = [x1, y1]` og `B = [x2, y2]`.
  Du henter ut *x*-koordinaten med `A[0]` og *y*-koordinaten med `A[1]`

```python
def distance(A, B):
    # Skriv kode som returnerer avstand her

```

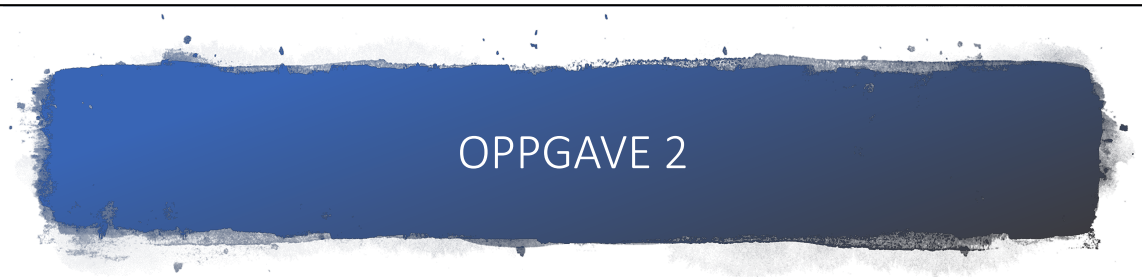
- Sjekk at koden faktisk fungerer

```python
A = [0,0]
B = [3,4]

print( distance(A,B) )
```

Lysbilde H-8

```
1  from pylab import *
2
3  def distance(A, B):
4      return sqrt( (B[0] - A[0])**2 + (B[1] - A[1])**2 )
5
6
7  A = [0,0]
8  B = [3,4]
9
10 print( distance(A,B) )
11
```

Lysbilde H-9



- Lag en funksjon `coulombforce(q, A, Q, B)` som regner ut den elektriske kraften som virker på ladning *q* i punkt *A* fra ladning *Q* i punkt *B*.
  Funksjonen skal returnere kraftens *x*- og *y*-komponent som en vektor (array).

```
36 def coulombforce(q,A,Q,B):
37     r = distance(A, B)
38     k = ###
39     F = ###
40     Fx = ###
41     Fy = ###
42     return asarray([Fx,Fy])
43
```

Lysbilde H-10

```
30 def coulombforce(q,A,Q,B):
31     r = distance(A, B)
32     k = 8.99E9
33     F = k*Q*q/r**2
34     Fx = F*(A[0]-B[0])/r
35     Fy = F*(A[1]-B[1])/r
36     return asarray([Fx,Fy])
37
```

Lysbilde H-11

- LAST NED KODEN `ladninger.py` FRA
  http://tinyurl.com/progfys2
- Denne koden er delvis ferdig, men en del må du fullføre. Til slutt skal du få en animasjon av partiklenes bevegelse.

Lysbilde H-12

OPPGAVE 3

- Legg inn funksjonene `distance()` og `coulombforce()` i `ladning.py`.

  Fyll ut konstanter og startverdier, slik at *q* er et positron (antielektron) i posisjon *A*(2, 2) og *Q* er et positron i posisjon *B*(0, 0)

Lysbilde H-13

```
from pylab import *
from tkinter import *
import time

#Denne koden lager et vindu for å vise animasjon
width = 600
height = 500
pxprcm = 38
window = Tk()
window.attributes("-topmost", True)
window.geometry(str(width)+"x"+str(height))
window.title("Simulering av elektrisk ladning")
c = Canvas(window, width = width, height = height, bg='white')
c.pack()
#Denne funksjonen plasserer et sirkelobjekt i posisjon P på lerrettet
def place(obj, P):
    xw = c.coords(obj)[2] - c.coords(obj)[0]
    yw = c.coords(obj)[3] - c.coords(obj)[1]
    c.coords(obj, width/2+P[0]*pxprcm, height/2-P[1]*pxprcm, width/2+P[0]*pxprcm+xw, height/2-P[1]*pxprcm+yw)



#Funksjon som regner ut avstand mellom to punkter A og B
def distance(A, B):
    return sqrt((A[0]-B[0])**2 + (A[1]-B[1])**2)


#Funksjon som regner ut kraften mellom to ladninger, q og Q i hhv. posisjon A og B
def coulombforce(q,A,Q,B):
    r = distance(A, B)
    k = 8.99E9
    F = k*Q*q/r**2
    Fx = F*(A[0]-B[0])/r
    Fy = F*(A[1]-B[1])/r
    return asarray([Fx,Fy])


#Elementærladningen
e = 1.6E-19

#Bevegelig ladning
q = 1*e
m = 9.11E-31
A = [2,2]
v = [0,0]

#Fastspikret ladning
Q = 1*e
B = [0,0]

```

Lysbilde H-14

OPPGAVE 4

- Animasjonen kjøres ved hjelp av en `while`-løkke, som regner ut ny kraft, akselerasjon, fart og posisjon etter `dt = 0.01` sekunder. Partikkelen flyttes til nye koordinater med funksjonen `place()`

  Bruk funksjonen `coulombforce()` til å regne ut kraften `F` på *q* inne i `while`-løkken.
  Regn også ut akselerasjonen `a`, farten `v` og posisjonen `s` ved å bruke Euler-metoden inne i `while`-løkken.

Lysbilde H-15

```
52
53 #Tegner to partikler med radius lik 15 pixler
54 particle1 = c.create_oval(0,0,15,15, fill='red')
55 particle2 = c.create_oval(0,0,15,15, fill='blue')
56
57 #Plasserer de to partiklene i startposisjonene sine
58 place(particle1,A)
59 place(particle2,B)
60
61 #Tidssteg i den numeriske beregningen
62 dt = 0.01
63
64 #Animasjonen stopper ikke før brukeren avslutter programmet
65 while True:
66     #Regn ut kraften her
67     F = coulombforce(q,A,Q,B)
68
69     #Regn ut akselerasjonen her
70     a = F/m
71
72     #Regn ut farten her
73     v = v + a*dt
74
75     #Regn ut posisjonen her
76     A = A + v*dt
77
78     #Plasser partikkelen og oppdater animasjonsvinduet
79     place(particle1,A)
80     time.sleep(dt)
81     window.update()
82
83
84 #Viser animasjonsvinduet
85 window.mainloop()
```

Lysbilde H-16

OPPGAVER

5. Kjør simuleringen.
   Velg ulike verdier for startposisjonen *A*, utgangsfarten *v* og ladningene *q* og *Q*.
   Sjekk at simuleringen oppfører seg som forventet, og rett eventuelle feil.
   Hva skjer når partiklene kolliderer. Hvordan kan det forklares?

6. Velg en startposisjon og startfart slik at den ene partikkelen går i en sirkelbane rundt den andre partikkelen. Kan du klare å lage en ellipsebane?

7. Utvid koden med en (eller flere) partikler i ro. Forsøk å få den bevegelige partikkelen til å svinge frem og tilbake. Kan du få partikkelen til å gå i en åttetallsbevegelse?

UTFORDRING: En ladning som akselererer, avgir egentlig elektromagnetisk stråling og vil dermed miste noe kinetisk energi. Søk etter en formel for hvor mye energi som går tapt, og forbedre simuleringen slik at den tar høyde for strålingstapet.

Lysbilde H-17

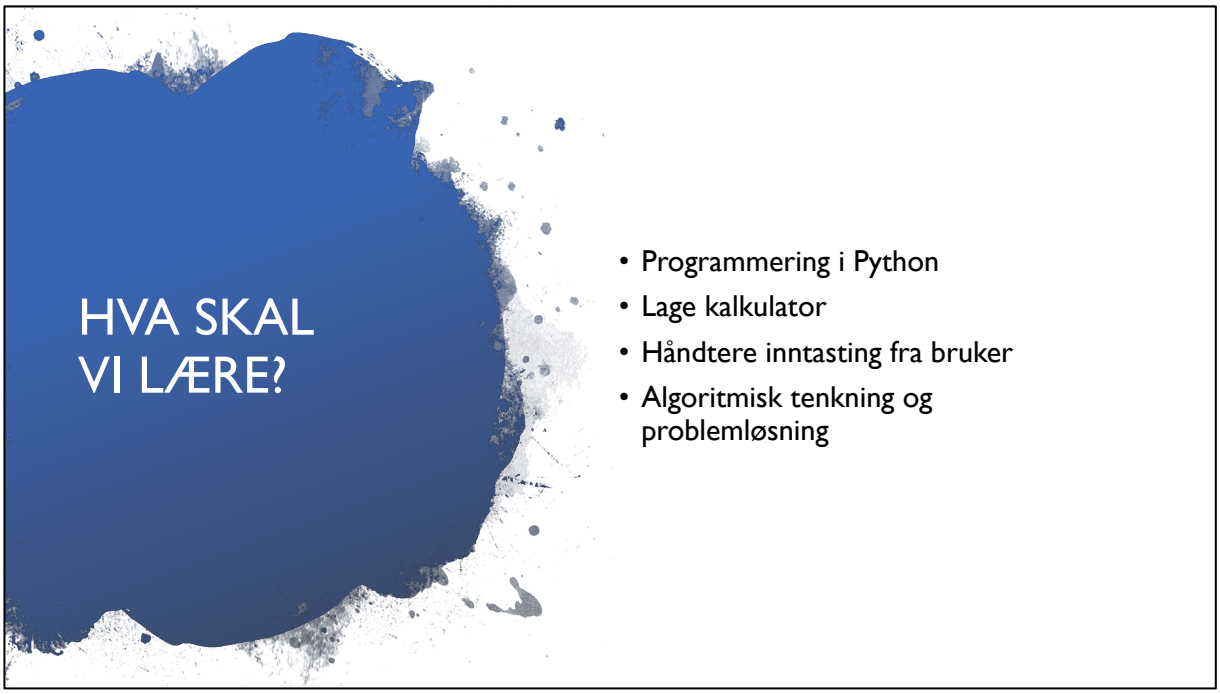
```python
from pylab import *
from tkinter import *
import time

#Denne koden lager et vindu for å vise animasjon
width = 600
height = 500
pxprcm = 38
window = Tk()
window.attributes("-topmost", True)
window.geometry(str(width)+"x"+str(height))
window.title("Simulering av elektrisk ladning")
c = Canvas(window, width = width, height = height, bg='white')
c.pack()
#Denne funksjonen plasserer et sirkelobjekt i posisjon P på lerrettet
def place(obj, P):
    xw = c.coords(obj)[2] - c.coords(obj)[0]
    yw = c.coords(obj)[3] - c.coords(obj)[1]
    c.coords(obj, width/2+P[0]*pxprcm, height/2-P[1]*pxprcm, width/2+P[0]*pxprcm+xw, height/2-P[1]*pxprcm+yw)



#Funksjon som regner ut avstand mellom to punkter A og B
def distance(A, B):
    return sqrt((A[0]-B[0])**2 + (A[1]-B[1])**2)


#Funksjon som regner ut kraften mellom to ladninger, q og Q i hhv. posisjon A og B
def coulombforce(q,A,Q,B):
    r = distance(A, B)
    k = 8.99E9
    F = k*Q*q/r**2
    Fx = F*(A[0]-B[0])/r
    Fy = F*(A[1]-B[1])/r
    return asarray([Fx,Fy])


#Elementærladningen
e = 1.6E-19

#Bevegelig ladning
q = -1*e
m = 9.11E-31
A = [0,0]
v = [15,10]

#Fastspikret ladning
Q = 1*e
B = [0,2]
C = [0,-2]

#Tegner partikler med radius lik 15 pixler
particle1 = c.create_oval(0,0,15,15, fill='red')
particle2 = c.create_oval(0,0,15,15, fill='blue')
particle3 = c.create_oval(0,0,15,15, fill='blue')

#Plasserer de partiklene i startposisjonene sine
place(particle1,A)
place(particle2,B)
place(particle3,C)

#Tidssteg i den numeriske beregningen
dt = 0.01


while True:
    # Regn ut kraften her
    F = coulombforce(q,A,Q,B) + coulombforce(q,A,Q,C)

    #Regn ut akselerasjonen her
    a = F/m

    #Regn ut farten her
    v = v + a*dt

    #Regn ut posisjonen her
    A = A + v*dt

    #Plasser partikkelen og oppdater animasjonsvinduet
    place(particle1,A)
    time.sleep(dt)
    window.update()


#Viser animasjonsvinduet
window.mainloop()

```

Lysbilde H-18

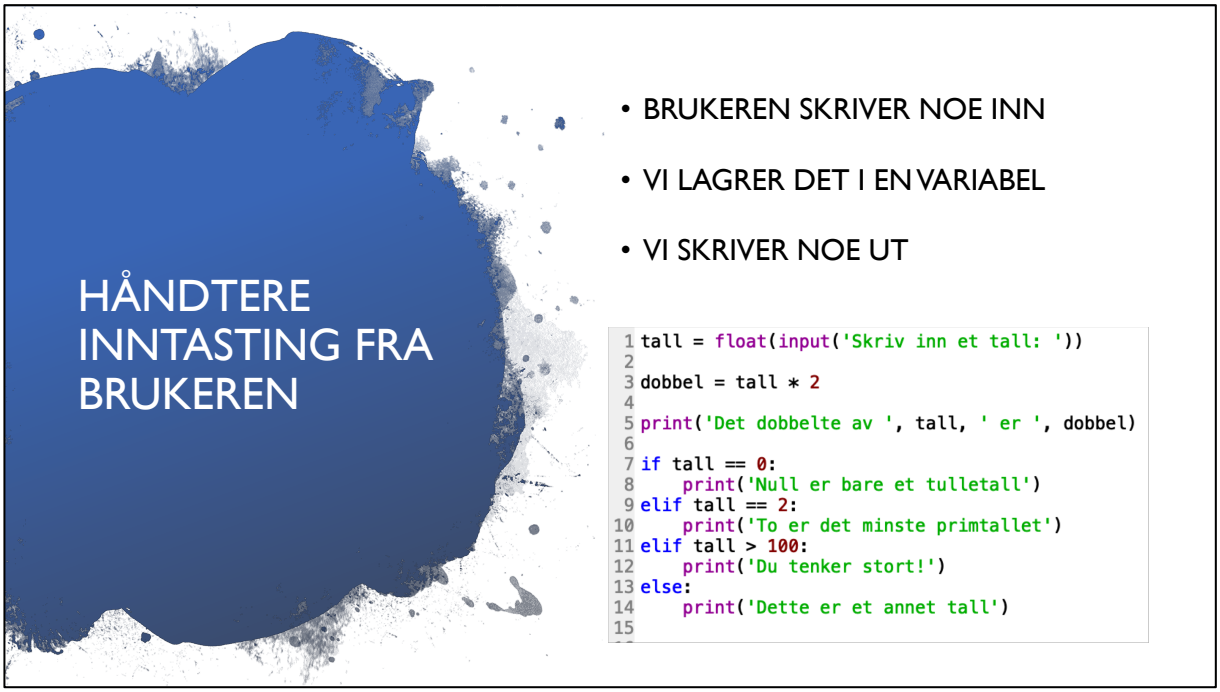
VEDLEGG I: Lysbildepresentasjon for opplegg 3

Lysbilde I-1

Lysbilde I-2



Lysbilde I-3



Lysbilde I-4

OPPGAVE

- LAG EN FOTONOMATOR
  - Programmet skal spørre brukeren om en bølgelengde.
  - Brukeren skal få tilbake frekvens, energi og farge.
  - Utvid programmet slik at det viser fargen med RGB-kode.

FARGE	NAVN	BØLGELENGDER (nm)	RGB-KODE
	Usynlig	> 700	(0, 0, 0)
	Rød	630 – 700	(1, 0, 0)
	Oransje	590 – 630	(1, 0.5, 0)
	Gul	560 – 590	(1, 1, 0)
	Grønn	520 – 560	(0, 1, 0)
	Turkis	490 – 520	(0, 1, 1)
	Blå	450 – 490	(0, 0, 1)
	Fiolett	400 – 450	(1, 0, 1)
	Usynlig	< 400	(0, 0, 0)

Lysbilde I-5

```
from pylab import *

# Simpel fotonfarge
def fargenavn(l):
    l = float(l)
    if l < 400:
        return('Usynlig lys')
    elif l < 450:
        return('Fiolett')
    elif l < 490:
        return('Blå')
    elif l < 520:
        return('Cyan')
    elif l < 560:
        return('Grønn')
    elif l < 590:
        return('Gul')
    elif l < 625:
        return('Oransje')
    elif l < 700:
        return('Rød')
    else:
        return('Usynlig lys')


#Avansert fotonfarge
def fotonfarge(l):

    l = float(l)
    #Fra usynlig til fiolett(rød+blå)
    if l >= 350 and l  <= 400:
        R = (l - 350)/(400 - 350)
        G = 0.0
        B = (l - 350)/(400 - 350)
    #Fiolett (rød+blå) fra 400 til 450
    #Fra fiolett (rød+blå) til blå
    elif l >= 400 and l <= 450:
        R = (l - 450)/(400-450)
        G = 0.0
        B = 1
    #Blå fra 450 til 490
    #Fra blå til cyan (blå+grønn)
    elif l >= 450 and l <= 490:
        R = 0.0
        G = (l - 450)/(490 - 450)
        B = 1.0
    #Cyan fra 490 til 520
    #Fra cyan (blå+grønn) til grønn
    elif l >= 490 and l <= 520:
        R = 0.0
        G = 1.0
        B = (l - 520)/(490 - 520);
    #Grønn fra 520 til 560
    #Fra grønn til gul (rød+grønn)
    elif l >= 520 and l <= 560:
        R = (l-520)/(560-520)
        G = 1.0
        B = 0
    #Gul fra 560 til 635
    #Fra gul (rød+grønn) til rød
    elif l >= 560 and l <= 635:
        R = 1.0
        G = (l-635)/(560-635)
        B = 0.0
    #Rødt fra 635 til 700
    #Fra rødt til usynlig
    elif l >= 625 and l <= 700:
        R = (l-700)/(625-700)
        G = 0.0
        B = 0.0
    #Usynlig lys
    else:
        R = 0
        G = 0
        B = 0
        print('Usynlig lys')
    return (R, G, B)


print('$$$$$$$$$$$$$$$$$')
print('   FOTONOMATOREN')
print('$$$$$$$$$$$$$$$$$')
inntast = float(input('Bølgelgende i nm: '))
while inntast != 0.0:
    print('\n\n\n\n\n\n\n')
    fig = figure()
    ax = fig.add_subplot(1, 1, 1)
    ax.set_facecolor(fotonfarge(inntast))
    show()
    f = 3E8/(inntast*1E-9)
    E = 6.63E-34*f
    print('FARGE:       ' + fargenavn(inntast))
    print('BØLGELENGDE: ' + str(inntast) + ' nm')
    print('FREKVENS:    ' + str(round(f/1E12,1)) + ' THz')
    print('ENERGI:      ' + str(round(E/1E-18,3)) + ' aJ')
    inntast = float(input('Skriv 0 for å avslutte.\nBølgelengde i nm: '))

print('\nTakk for i dag! 👋🏻🏻')
```

Lysbilde I-6

VEDLEGG J: Datafiler og programkode

Her følger programkode og datafiler som ble utdelt til elevene, samt ferdig kode som løsningsforslag til oppgavene. Disse filene er også tilgjengelige for nedlastning på nett fra http://tinyurl.com/progfys1 (opplegg 1 og 3) og http://tinyurl.com/progfys2 (opplegg 2). På disse nettadressene finnes også oppgavearkene, lysbildepresentasjonene og noen andre nyttige programkoder.

Warholm400.txt (Datafil til elevene, opplegg 1)

```
# Dataene viser tidspunktet Warholm hoppet over en hekk i et 400-meterløp i London 2018
# Strekning(m) Tid(s)
0.0 0.0
45.0 5.5
80.0 9.2
115.0 12.9
150.0 16.8
185.0 20.6
220.0 24.7
255.0 28.9
290.0 33.0
325.0 37.5
360.0 42.0
400.0 47.65
```

Løpetur.txt (Datafil til elevene, opplegg 1)

```
1.    # Dataene viser hvor mange hele minutter og sekunder læreren ville brukt per
      kilometer med farten han hadde ved hvert tidspunkt. Etter 50 sekunder ville læreren
      f.eks. brukt 9 minutter og 57 sekunder på å løpe en kilometer, mens etter 100
      sekunder ville læreren brukt 5 minutter og 52 sekunder per kilometer.
2.    # Tid(s) Fart(min/km) (sek/km)
3.    0 9 57
4.    10 10 00
5.    20 10 03
6.    30 10 06
7.    40 10 09
8.    50 9 57
9.    60 8 38
10.   70 7 53
11.   80 7 02
12.   90 6 25
13.   100 5 52
14.   110 5 33
15.   120 5 28
16.   130 5 23
17.   140 5 23
18.   150 5 40
19.   160 5 59
20.   170 6 20
21.   180 6 31
22.   190 6 21
23.   200 6 12
24.   210 6 03
25.   220 5 53
26.   230 5 47
27.   240 5 33
28.   250 5 48
29.   260 5 53
30.   270 5 58
31.   280 6 03
32.   290 6 08
33.   300 6 15
34.   310 6 19
35.   320 6 24
36.   330 6 32
37.   340 6 37
38.   350 6 43
39.   360 6 49
40.   370 6 55
41.   380 7 02
42.   390 7 10
43.   400 7 15
44.   410 7 23
45.   420 7 31
46.   430 7 38
47.   440 7 48
48.   450 7 58
49.   460 8 05
50.   470 8 15
51.   480 8 25
52.   490 8 15
53.   500 8 04
54.   510 7 53
55.   520 7 42
56.   530 7 33
57.   540 7 23
58.   550 7 14
59.   560 7 06
60.   570 7 13
```

```
  580 7 21
  590 7 28
  600 7 37
  610 7 46
  620 7 51
  630 6 12
  640 5 09
  650 5 15
  660 5 23
  670 5 30
  680 5 38
  690 5 46
  700 5 54
  710 5 59
  720 5 49
  730 5 41
  740 5 33
  750 5 30
  760 5 43
  770 5 55
  780 6 09
  790 6 26
  800 6 42
  810 7 02
  820 7 25
  830 7 43
  840 8 07
  850 8 33
  860 9 04
  870 9 40
  880 9 27
  890 8 43
  900 8 04
  910 7 30
  920 7 04
  930 6 34
  940 6 07
  950 5 51
  960 5 41
  970 5 39
  980 5 37
  990 5 35
 1000 5 44
 1010 6 05
 1020 6 24
 1030 6 35
 1040 6 38
 1050 6 41
 1060 6 45
 1070 6 48
 1080 6 52
 1090 6 55
 1100 6 59
 1110 7 02
 1120 7 06
 1130 7 11
 1140 7 14
 1150 7 10
 1160 7 05
 1170 7 01
 1180 6 55
 1190 6 51
 1200 6 46
 1210 6 41
 1220 6 38
 1230 6 34
```

```
127.  1240 6 30
128.  1250 6 26
129.  1260 6 22
130.  1270 6 18
131.  1280 6 15
132.  1290 6 11
133.  1300 6 07
134.  1310 6 03
135.  1320 6 00
136.  1330 5 56
137.  1340 5 53
138.  1350 5 50
139.  1360 5 47
140.  1370 5 45
141.  1380 5 48
142.  1390 5 50
143.  1400 5 53
144.  1410 5 54
145.  1420 5 57
146.  1430 5 59
147.  1440 6 02
148.  1450 6 05
149.  1460 6 07
150.  1470 6 10
151.  1480 6 13
152.  1490 6 15
153.  1500 6 18
154.  1510 6 21
155.  1520 6 23
156.  1530 6 27
157.  1540 6 30
158.  1550 6 32
159.  1560 6 26
160.  1570 6 16
161.  1580 6 05
162.  1590 5 55
163.  1600 5 46
164.  1610 5 36
165.  1620 5 27
166.  1630 5 25
167.  1640 5 25
168.  1650 5 25
169.  1660 5 25
170.  1670 5 25
171.  1680 5 25
172.  1690 5 25
173.  1700 5 25
```

Løsningsforslag: Warholm400.py (opplegg 1)

```
1.  from pylab import *
2.
3.  # Laste inn data fra Warholms løp
4.  data = loadtxt("Warholm400.txt", delimiter=" ")
5.  posisjon = data[:,0]
6.  tid = data[:,1]
7.  n = len(tid)
8.
9.  #Tegne grafen til tid/posisjon
10. plot(tid, posisjon, 'ro-')
11. xlabel('Tid i sekunder')
12. ylabel('Strekning i meter')
13. show()
14.
15. #Beregne fart
16. fart = zeros(n-1)
17. for i in range(n-1):
18.     fart[i] = (posisjon[i+1]-posisjon[i])/(tid[i+1]-tid[i])
19.
20. #Tegne grafen til tid/fart
21. plot(tid[0:-1], fart, 'bo-')
22. xlabel('Tid i sekunder')
23. ylabel('Fart i m/s')
24. show()
25.
26. #Beregne akselerasjon
27. akselerasjon = zeros(n-2)
28. for i in range(n-2):
29.     akselerasjon[i] = (fart[i+1]-fart[i])/(tid[i+1]-tid[i])
30.
31. #Tegne grafen til tid/akselerasjon
32. plot(tid[0:-2], akselerasjon, 'go-')
33. xlabel('Tid i sekunder')
34. ylabel('Akselerasjon i $m/s^2$')
35. show()
```

Løsningsforslag: Løpetur.py (opplegg 1)

```
1.  from pylab import *
2.
3.  #Henter data fra lærerens løpetur og antall datapunkter
4.  data = loadtxt("løpetur.txt", delimiter=" ")
5.  n = len(data)
6.  dt = 10
7.
8.  #Deler opp data, kolonne 0 er tid, kolonne 1 er minutter pr km, kolonne 2 er sekunde
    r pr km
9.  tid = data[:,0]
10. minutter = data[:,1]
11. sekunder = data[:,2]
12.
13. #Regner fra tid pr km til fart i m/s
14. fart = 1000/(minutter*60+sekunder)
15.
16. #Tegne grafen over fart og tid (i minutter)
17. plot(tid/60,fart)
18. xlabel('Tid i minutter')
19. ylabel('Fart i m/s')
20. show()
21.
22. #Lage tomme lister for strekning og akselerasjon
23. s = zeros(n)
24. a = zeros(n)
25.
26. #Beregne strekning med ny = gammel + endring
27. #Beregne akselersajon med derivert = endring / tid
28. for i in range(len(tid)-1):
29.     s[i+1] = s[i] + fart[i] * dt
30.     a[i+1] = (fart[i+1]-fart[i])/dt
31.
32. #Tegne grafen til strekning i km
33. plot(tid/60,s/1000)
34. xlabel('Tid i minutter')
35. ylabel('Strekning i km')
36. show()
37.
38. #Tegne grafen til akselerasjon i m/s^2
39. plot(tid/60,a)
40. xlabel('Tid i minutter')
41. ylabel('Akselerasjon i $m/s^2$')
42. show()
43.
44. print('Læreren løp ' + str(round(s[-1]/1000,1)) + ' km til sammen')
```

Ladning.py (Delvis ferdig kode til elevene, opplegg 2)

```
1.  from pylab import *
2.  from tkinter import *
3.  import time
4.
5.
6.
7.
8.  #Denne koden lager et vindu for å vise animasjon
9.  width = 600
10. height = 500
11. pxprcm = 38
12. window = Tk()
13. window.attributes("-topmost", True)
14. window.geometry(str(width)+"x"+str(height))
15. window.title("Simulering av elektrisk ladning")
16. c = Canvas(window, width = width, height = height, bg='white')
17. c.pack()
18. #Denne funksjonen plasserer et sirkelobjekt i posisjon P på lerrettet
19. def place(obj, P):
20.     xw = c.coords(obj)[2] - c.coords(obj)[0]
21.     yw = c.coords(obj)[3] - c.coords(obj)[1]
22.     c.coords(obj, width/2+P[0]*pxprcm, height/2-
    P[1]*pxprcm, width/2+P[0]*pxprcm+xw, height/2-P[1]*pxprcm+yw)
23.
24.
25.
26.
27. #Funksjon som regner ut avstand mellom to punkter A og B
28. def distance(A, B):
29.     # !!! SKRIV KODE FOR Å REGNE UT OG RETURNERE AVSTANDEN HER !!!
30.
31.
32.
33.
34.
35. #Funksjon som regner ut kraften mellom to ladninger, q og Q i hhv. posisjon A og B
36. def coulombforce(q,A,Q,B):
37.     r = distance(A, B)
38.     k = ###
39.     F = ###
40.     Fx = ###
41.     Fy = ###
42.     return asarray([Fx,Fy])
43.
44.
45. # !!! SETT RIKTIGE VERDIER TIL KONSTANTENE HER !!!
46. #Elementærladningen
47. e = ###
48.
49.
50. #Bevegelig ladning
51. q = ###
52. m = ###
53. A = [2,0]
54. v = [0,0]
55.
56. #Spirket ladning
57. Q = ###
58. B = [0,0]
59.
60.
```

```
#Tegner to partikler med radius lik 15 pixler
particle1 = c.create_oval(0,0,15,15, fill='red')
particle2 = c.create_oval(0,0,15,15, fill='blue')

#Plasserer de to partiklene i startposisjonene sine
place(particle1,A)
place(particle2,B)

#Tidssteg i den numeriske beregningen
dt = 0.01

#Animasjonen stopper ikke før brukeren avslutter programmet
while True:
    #Regn ut kraften her
    F = ###

    #Regn ut akselerasjonen her
    a = ###

    #Regn ut farten her
    v = ###

    #Regn ut posisjonen her
    A = ###

    #Plasser partikkelen og oppdater animasjons-vinduet
    place(particle1,A)
    time.sleep(dt)
    window.update()


#Viser animasjonsvinduet
window.mainloop()
```

Løsningsforslag: Ladning.py (opplegg 2)

```
1.  from pylab import *
2.  from tkinter import *
3.  import time
4.
5.  #Lager et vindu for animasjonen
6.  width = 600
7.  height = 500
8.  pxprcm = 38
9.  window = Tk()
10. window.attributes("-topmost", True)
11. window.geometry(str(width)+"x"+str(height))
12. window.title("Simulering av elektrisk ladning")
13. c = Canvas(window, width = width, height = height, bg='white')
14. c.pack()
15.
16. #Regn ut kraften mellom to ladninger
17. def coloumbforce(q,Q,A,B):
18.     r = distance(A, B)
19.     k = 8.9877E9
20.     F = k*Q*q/r**2
21.     Fx = F*(A[0]-B[0])/r
22.     Fy = F*(A[1]-B[1])/r
23.     return asarray([Fx,Fy])
24.
25. #Regn ut avstand mellom to punkter
26. def distance(A, B):
27.     return sqrt((A[0]-B[0])**2 + (A[1]-B[1])**2)
28.
29.
30. #Plassere et sirkelobjekt på gitte koordinater
31. def place(obj, P):
32.     c.coords(obj, width/2+P[0]*pxprcm, height/2-
    P[1]*pxprcm, width/2+P[0]*pxprcm+15, height/2-P[1]*pxprcm+15)
33.
34. #Elektronladningen
35. e = 1.6E-19
36.
37.
38. #Bevegelig ladning i feltet av Q
39. q = -1*e
40. m = 9.11E-31
41. A = [0,0]
42. v = [12, 0]
43.
44. #Spirket ladning
45. Q = 1*e
46. B = [0,-2]
47. #C = [0,2]
48.
49. #Tegner to partikler med radius lik 10 pixler
50. particle1 = c.create_oval(0,0,10,10, fill='red')
51. particle2 = c.create_oval(0,0,10,10, fill='blue')
52. #particle3 = c.create_oval(0,0,10,10, fill='blue')
53.
54. #Plasserer de to partiklene i startposisjonene sine
55. place(particle1,A)
56. place(particle2,B)
57. #place(particle3,C)
58.
59. #Tidssteg i den numeriske beregningen
60. dt = 0.01
```

```
while True:
    # Regn ut kraften her
    F = coloumbforce(q,Q,A,B) #+ coloumbforce(q,Q,A,C)

    #Regn ut akselerasjonen her
    a = F/m

    #Regn ut farten her
    v = v + a*dt

    #Regn ut posisjonen her
    A = A + v*dt

    #Plasser partikkelen og oppdater vinduet
    place(particle1,A)
    time.sleep(dt)
    window.update()


#Animasjonsvinduet
window.mainloop()
```

Løsningsforslag: Fotonomator.py (opplegg 3)

```
1.  from pylab import *
2.
3.  #Simpel fotonfarge
4.  def fargenavn(l):
5.      l = float(l)
6.      if l < 400:
7.          return('Usynlig lys')
8.      elif l < 450:
9.          return('Fiolett')
10.     elif l < 490:
11.         return('Blå')
12.     elif l < 520:
13.         return('Cyan')
14.     elif l < 560:
15.         return('Grønn')
16.     elif l < 590:
17.         return('Gul')
18.     elif l < 625:
19.         return('Oransje')
20.     elif l < 700:
21.         return('Rød')
22.     else:
23.         return('Usynlig lys')
24.
25.
26. #Avansert fotonfarge
27. def fotonfarge(l):
28.
29.     l = float(l)
30.     #Fra usynlig til fiolett(rød+blå)
31.     if l >= 350 and l  <= 400:
32.         R = (l - 350)/(400 - 350)
33.         G = 0.0
34.         B = (l - 350)/(400 - 350)
35.     #Fiolett (rød+blå) fra 400 til 450
36.     #Fra fiolett (rød+blå) til blå
37.     elif l >= 400 and l <= 450:
38.         R = (l - 450)/(400-450)
39.         G = 0.0
40.         B = 1
41.     #Blå fra 450 til 490
42.     #Fra blå til cyan (blå+grønn)
43.     elif l >= 450 and l <= 490:
44.         R = 0.0
45.         G = (l - 450)/(490 - 450)
46.         B = 1.0
47.     #Cyan fra 490 til 520
48.     #Fra cyan (blå+grønn) til grønn
49.     elif l >= 490 and l <= 520:
50.         R = 0.0
51.         G = 1.0
52.         B = (l - 520)/(490 - 520);
53.     #Grønn fra 520 til 560
54.     #Fra grønn til gul (rød+grønn)
55.     elif l >= 520 and l <= 560:
56.         R = (l-520)/(560-520)
57.         G = 1.0
58.         B = 0
59.     #Gul fra 560 til 635
60.     #Fra gul (rød+grønn) til rød
61.     elif l >= 560 and l <= 635:
62.         R = 1.0
63.         G = (l-635)/(560-635)
```

```
        B = 0.0
    #Rødt fra 635 til 700
    #Fra rødt til usynlig
    elif l >= 625 and l <= 700:
        R = (l-700)/(625-700)
        G = 0.0
        B = 0.0
    #Usynlig lys
    else:
        R = 0
        G = 0
        B = 0
        print('Usynlig lys')
    return (R, G, B)


print('⚡⚡⚡⚡⚡⚡⚡⚡⚡⚡⚡⚡⚡⚡⚡⚡⚡⚡⚡')
print('   FOTONOMATOREN')
print('⚡⚡⚡⚡⚡⚡⚡⚡⚡⚡⚡⚡⚡⚡⚡⚡⚡⚡⚡')
inntast = float(input('Bølgelgende i nm: '))
while inntast != 0.0:
    print('\n\n\n\n\n\n')
    fig = figure()
    ax = fig.add_subplot(1, 1, 1)
    ax.set_facecolor(fotonfarge(inntast))
    show()
    f = 3E8/(inntast*1E-9)
    E = 6.63E-34*f
    print('FARGE:       ' + fargenavn(inntast))
    print('BØLGELENGDE: ' + str(inntast) + ' nm')
    print('FREKVENS:    ' + str(round(f/1E12,1)) + ' THz')
    print('ENERGI:      ' + str(round(E/1E-18,3)) + ' aJ')
    inntast = float(input('Skriv 0 for å avslutte.\nBølgelengde i nm: '))

print('\nTakk for i dag! 👋🤓')
```

Sigurd Torp Nordby

Programmering og algoritmisk tenkning i fysikkundervisning

NTNU
Kunnskap for en bedre verden