Bacheloroppgave

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

Sara Mohammadi

Anvendelse av programmering i matematikk

Matematisk modellering, algoritmer og programkoder som støtter matematikklærere i programmerings undervisning på den norske videregående skolen

Bacheloroppgave i ingeniørfag, data
Veileder: Majid Rouhani

Mai 2021

NTNU
Kunnskap for en bedre verden

Sara Mohammadi

Anvendelse av programmering i matematikk

Matematisk modellering, algoritmer og programkoder som støtter matematikklærere i programmerings undervisning på den norske videregående skolen

Bacheloroppgave i ingeniørfag, data
Veileder: Majid Rouhani
Mai 2021

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU

Kunnskap for en bedre verden

Forord

Jeg har alltid vært interessert i matematikk helt fra da jeg var på barneskolen til nå som er i slutten av min vei for å bli dataingeniør. Etter at jeg begynte med dette studiet, virket programmering svært spennende for meg. Den har også mye i felles med matematikk; for begge trenger man å tenke logisk for å finne en løsning.

Grunnen til at jeg har valgt denne oppgaven, er at den handler om både matematikk og programmering noe som gjør oppgaven mer interessant og engasjerende. Resultatet av dette prosjektet skal være et hjelpemiddel for matematikklærere på den norske videregående skolen. Dette var noe som motiverte meg hele tiden til å jobbe så best og effektivt som mulig for at det endelige arbeidet skal være til nytte i fremtiden.

For å løse oppgaven begynte jeg først med å forske om flere parametere som virket til å være viktig for både å sette meg i gang, og å gå videre i prosessen. De mest avgjørende parameterne var å kjenne til miljøet der problemet ble reist, og utforske eksisterende løsninger for problemet. Videre jobbet jeg med å samle inn en del matematiske oppgaver som regnes som interessante, motiverende, og viktige oppgaver, samtidig som de lar seg å programmeres. Det måtte også tas hensyn til at oppgavene er i samsvar med de nye læreplaner for matematikkfaget. Resultatet er altså en samling av klassifiserte matematiske oppgaver og deres løsninger. Løsningene inkluderer algoritmer for koding av oppgaven i Python samt selve programkoder.

Jeg vil til slutt takke min veileder Majid Rouhani og oppdragsgiver NTNU-AIT for god veiledning og kommunikasjon gjennom hele prosjektet.

Sara Mohammadi

Trondheim, 20. mai 2021

When we say we educate
children it sounds like
something we do to them.
That's not the way it happens,
we don't educate them. We
create contexts in which they
will learn.

(PAPERT, SITERT I PEASE, 1989)

Oppgavetekst

Opprinnelig oppgavetekst var «Programmering i STEM-fag: Matematisk modellering, algoritmer og programkode som støtter lærere i programmerings undervisning».

Hensikten med oppgaven var å finne ut hvilke fenomener som kan programmeres i STEM-fag og anbefalte framgangsmåter for implementasjon av disse.

Oppgaveforslag var å finne:

- Hvilke fenomener kan programmeres i STEM-fag?
- Hva er anbefalte framgangsmåter for implementasjon av disse fenomenene?

Den opprinnelige tittelen er nå byttet ut til «Anvendelse av programmering i matematikk: Matematisk modellering, algoritmer og programkoder som støtter lærere i programmerings undervisning på den norske videregående skolen».

Grunnen for begrensning av oppgaven er at oppgavens tittel dekker flere fagområder, og det er svært krevende å jobbe med alle av de fagene gjennom en bacheloroppgave. Dermed er oppgaven begrenset til bare matematikkfaget på videregående skole.

Ønsket med oppgaven er å utforske matematiske oppgaver som egner seg best for å øke dybdeforståelse i matematikk og elevenes interesse innen programmering, og å utvikle algoritmer og programkoder for de utvalgte oppgaver for å støtte lærere til å anvende programmering i matematikkfaget.

Sammendrag

I 2019 ble endringer i læreplaner for grunnskole og videregående skole vedtatt av Kunnskapsdepartementet, og de nye læreplanene ble tatt i bruk fra og med skolestart høsten 2020. Ifølge Kunnskapsdepartementet (2019), med de nye læreplanene får skolen et verdiløft. Dette blir den største endringen av skolens innhold siden Kunnskapsløftet i 2006. Endringer i de nye læreplanene skal bidra til at elever får et bedre grunnlag for kritisk tenkning, utforskning og kreativitet.

Som følge av disse endringene, skal lærere sette fokus på begreper som algoritmisk tenkning, problemløsnings strategier og programmering (Stenlund, 2020). Disse gjelder spesielt matematikkundervisning (Utdanningsdirektoratet, 2020).

I forbindelse med fagfornyelsen spiller matematikkfaget en nøkkelrolle i programmerings opplæring, på grunn av programmering benytter matematikk på mange måter, og i flere områder (Bueie, 2019).

Innføringen av programmering i skolen har skapt noen utfordringer hos lærere i tjenesten (Stenlund, 2020). Lærernes behov for programmerings kunnskap, har ført til at flere universiteter og høyskoler startet å tilby etterutdannings opplæring i programmering for lærere. Lærere har også vært aktive til å delta i slike kurs for å lære å anvende programmering i sine fag.

Til tross for slike tilbud, er fremdeles en utfordring for lærere å integrere programmering i deres undervisnings plan. Det ser derfor nødvendig ut for en klassifisert samling av matematiske oppgaver som er også i samsvar med de nye læreplanene for matematikkfaget på VSG.

For å løse problemet er et Design Science Research (DSR) prosjekt gjennomført. Innsatsen med prosjektet er å utforske anvendelse av programmering i matematikk, og utvikle matematiske algoritmer og programkoder for å lage en klassifisert samling av matematiske oppgaver som hjelpemiddel for matematikklærere i tjenesten.

Som resultat av prosjektet er til sammen 37 oppgaver samlet inn under et hefte. Disse oppgavene tilhører matematikkfagene 1P, 1T, 2P, R1, R2, S1 og S2 på VGS. For enhver oppgave utarbeides løsnings strategien, algoritmen, og programkoden. Algoritmen i hver oppgave viser skrittvis hvordan problemet kan programmeres.

Hoved temaer i de valgte oppgavene er blant de temaene som er understreket i de nye læreplanene som for eksempel problemløsning, algoritmisk tenkning, og programmering. Det er også forsøkt å velge de oppgavene som regnes som interessante, engasjerende og viktige matematiske oppgaver med tanke på å stimulere kreativitet hos elevene, og hjelpe dem til å bli gode problemløsere.

Abstract

In 2019, changes in curricula for primary and secondary school were adopted by the Ministry of Education, and the new curricula were implemented from the start of school in the autumn of 2020. According to the Ministry of Education, with the new curricula, the school gets a boost in value. This will be the largest change in the school's content since the Knowledge Promotion in 2006. Changes in the new curricula will help pupils to have a better basis for critical thinking, exploration, and creativity (Kunnskapsdepartementet, 2019).

As a result of these changes, teachers will focus on concepts such as algorithmic thinking, problem-solving strategies, and programming (Stenlund, 2020). These are especially applied to mathematics teaching (Utdanningsdirektoratet, 2020).

In connection with the education reform, the subject of mathematics plays a key role in programming education, due to programming uses mathematics in many ways, and in several areas (Bueie, 2019).

The introduction of programming in schools has created some challenges for in-service teachers (Stenlund, 2020). The teachers' need for programming knowledge has led to several universities and colleges starting to offer continuing education courses in programming for teachers. Teachers have also been active in attending such courses to learn how to apply programming in their subjects.

Despite such offerings, it is still a challenge for teachers to integrate programming into their curriculum. Therefore, it seems necessary for a classified collection of mathematical exercises that is also in accordance with the new curricula for the mathematics subject at Norwegian upper secondary schools (USS).

To solve the problem, a Design Science Research (DSR) project has been completed. The effort with the project is to search the use of programming in mathematics, and develop mathematical algorithms and program codes to create a classified collection of mathematical problems as an aid for mathematics teachers in the service.

As a result of the project, a total of 37 exercises have been collected under a booklet. These exercises belong to the mathematics subjects 1P, 1T, 2P, R1, R2, S1 and S2 at UUS. For each task, the solution strategy, algorithm, and program code are prepared. The algorithm in each task shows step by step how the problem can be programmed.

The main subjects in the selected exercises are among the subjects that are emphasized in the new curricula, such as problem-solving, algorithmic thinking, and programming. It is also attempted to select the tasks that are considered interesting, engaging, and important mathematical tasks with a view to, stimulating the creativity of the pupils, and helping them to become good problem-solvers.

Innhold

Figurer

Tabeller

Programmer

Akronymer og forkortelser

AIT: Anvendt informasjonsteknologi

AT: Algoritmisk tenking

CAS: Computer Algebra System

CT: Computational thinking

DSR: Design Science Research

IKT: Informasjon og Kommunikasjonsteknologi

IS: Information Systems

IT: Information Technology

KD: Kunnskapsdepartementet

LK06: Kunnskapsløftet 2006

NOU: Norges offentlige utredninger

STEM: Science, Technology, Engineering, and Mathematics

UDIR: Utdanningsdirektoratet

USS: Upper Secondary Schools

VGS: Videregående skole

Viktige begreper

Algoritme: "En algoritme er en metode for å løse et problem som består av nøyaktig definerte instruksjoner"(Futschek, 2006, s. 160).

Algoritmisk tenkning: Algoritmisk tenkning er en problemløsnings prosess hvor man tenker som en informatiker når man skal løse et problem (Sevik mfl., 2018).

Dybdelæring: "Det å gradvis utvikle kunnskap og varig forståelse av begreper, metoder og sammenhenger i fag og mellom fagområder. Det innebærer at vi reflekterer over egen læring og bruker det vi har lært på ulike måter i kjente og ukjente situasjoner, alene eller sammen med andre"(Utdanningsdirektoratet, 2019).

Fagfornyelsen: "Navnet på arbeidet med de nye læreplaner som er tatt i bruk i grunnopplæringen fra og med skoleåret 2020"(Kunnskapsdepartementet, 2019).

Kjerneelementene: "Kjerneelementene er det viktigste og mest sentrale elevene skal lære i hvert fag, og gir retning og prioriteringer for de nye læreplanene som skal lages"(Kunnskapsdepartementet, 2018a).

Kompetanse: "Kompetanse betyr å kunne mestre utfordringer og løse oppgaver i ulike sammenhenger og omfatter både kognitiv, praktisk, sosial og emosjonell læring og utvikling, inkludert holdninger, verdier og etiske vurderinger"(Ludvigsenutvalget, 2015, s. 14) sitert i (Jakobsen, 2019, s. 20).

Kunnskapsløftet: "Kunnskapsløftet er en skolereform i grunnskolen og videregående opplæring vedtatt av Stortinget i juni 2004. Reformen ble innført fra høsten 2006 og førte til endringer i læreplaner, vurderings systemer og struktur"(SNL, 2020).

Læreplanverket: "Læreplanverket er det sentrale virkemiddelet for nasjonal styring av opplæringens innhold, og er den viktigste beskrivelsen av hvilken kompetanse elevene skal utvikle. Det har status som forskrift, definerer fagenes struktur og innhold og danner fundamentet for skolenes planlegging og gjennomføring av opplæringen"(Kunnskapsdepartementet, 2016, s. 9-10).

Problemløsning: "problemløsning er prosessen med å bevege seg mot et mål når veien til målet er usikker"(Martinez, 1998, s. 605).

Programmering: "Programmering går ut på å sette opp en serie instruksjoner som styrer maskinen og som avgjør hvordan den skal reagere på inndata, inntastinger, musebevegelser og annet"(Rossen, 2019).

Pseudokode: "Pseudokode er en algoritmisk metode som bruker naturlig språk for å gi in-depth, cross-platform kode beskrivelse"(Alhefdhi mfl., 2018).

1 Introduksjon og relevans

1.1 Bakgrunn

Samfunnet er i rask endring og stor tilgang på digitale ressurser i skolen og i samfunnet fører til økt digitalisering fremover. Følgelig blir kunnskap om hvordan digital teknologi fungerer vesentlig, og mennesker vil stadig ha behov for ny kompetanse og ferdigheter for å møte utfordringer ved å leve i et digitalt samfunn (Jakobsen, 2019).

I følge Balanskat og Engelhardt (2015) vil elever i fremtiden bidra til å utvikle teknologi, og de trenger utdanning som gjør dem i stand til dette. I denne forbindelsen har programmering blitt en viktig ferdighet i 21. århundre, og dermed forsøker myndighetene å tilpasse utdanningssektoren til å møte fremtidige samfunnskrav.

For at elever har en effektiv deltakelse i den digitale verden, har programmering blitt kjent som en grunnleggende ferdighet, og i det siste tiåret har det vært en økende interesse i å innføre programmering som skolefag. Mange land har allerede innlemmet programmering i skolens læreplaner for å lære elevene ferdigheter som problemløsning og logisk tenkning som regnes som viktige ferdigheter i dagens digitale samfunn (Forsström & Kaufmann, 2018).

Det har vært debatt om programmering skal være en del av Informasjon og Kommunikasjonsteknologi (IKT) eller om den skal innlemmes i hele læreplanen. De fleste skoler har innpasset programmering i andre fag, og i særdeleshet i matematikk (Forsström & Kaufmann, 2018). Finland og Sverige, for eksempel, har begge integrert programmering i matematikkfaget (Bocconi mfl., 2018) med tanke på at programmering fostrer ferdigheter som problemløsning og logisk tenkning, og motiverer elevene til å lære matematikk (Forsström & Kaufmann, 2018).

Norge begynte å revidere læreplaner, med formålet å møte utfordringene som ligger i omfattende og raske endringer i samfunnet. Kunnskapsdepartementet publiserte en digitaliserings strategi for primær, sekundær og yrkesfaglig utdanning for 2017-2021 der CT (Computational thinking som er oversatt som algoritmisk tenkning (AT) på norsk) og programmering burde innlemmes i læreplanene (Bocconi mfl., 2018).

I 2020 innførte Utdanningsdirektoratet programmering i det reviderte versjonen av læreplanen hvor programmering er innlemmet i, blant annet matematikkfaget (Bocconi mfl., 2018). I følge Bocconi mfl. (2018), er det imidlertid behov for hvordan programmering kan knyttes til andre fagområder, og hvordan programmering kan påvirke elevenes prestasjoner. Det er også behov for pedagogiske løsninger for implementering av programmering ved hjelp av en rekke verktøy og vurderinger. I denne sammenheng blir rollen til lærer fremhevet, og viktig å vurderes. Ved integrering av programmering i læreplan til matematikkfaget, kan lærerrollen bli utfordrende siden matematikklærere har kanskje ikke hatt forkunnskap om programmering (Forsström & Kaufmann, 2018). Det er riktig at ny teknologi og programvare gir matematikklærere nye muligheter i undervisningen, men slike endringer har også utfordringer som ikke fantes før (Jakobsen, 2019).

“Programmering er prosessen knyttet til utvikling og implementering av instruksjoner for dataprogrammer slik at datamaskinen kan utføre spesifikke oppgaver, løse problemer og støtte menneskelige interaksjoner”(Forsström & Kaufmann, 2018, s. 19).

Denne prosessen krever kunnskap om programmeringsspråk, ekspertise i relaterte fag til å utvikle algoritmer og å oppdage logikk, og evne til å analysere, forstå og løse problemer ved å verifisere algoritmiske krav og vurdere korrekthet og implementering av algoritmen i et bestemt programmeringsspråk (Forsström & Kaufmann, 2018).

1.1.1 Kunnskapsløftet

"I 2004 la departementet frem Stortingsmelding nr. 30 (2003–2004) «Kultur for læring» (Kunnskapsdepartementet, 2004), som presenterte utdanningsreformen Kunnskapsløftet. Målet med reformen var å gjøre elever og lærlinger bedre i stand til å møte kunnskapssamfunnets utfordringer"(Kunnskapsdepartementet, 2016, s. 9), og i skoleåret 2006-2007 ble kunnskapsløftet innført.

Gjennom «Læreplanverket for Kunnskapsløftet» også kalt LK06 ble fem grunnleggende ferdigheter innført (Kunnskapsdepartementet, 2006):

- Å kunne uttrykke seg muntlig
- Å kunne lese
- Å kunne regne
- Å kunne skrive
- Å kunne bruke digitale verktøy

Læreplaner ble også innført slik at det ble lagt vekt på hva elvene skulle lære, og ikke hva de skulle gjøre, og dette er videreført i fagfornyelsen (Kunnskapsdepartementet, 2019).

I læreplaner for matematikkfaget vil digitale ferdigheter bety å bruke digitale verktøy til læring, for eksempel utforsking, visualisering og presentasjon. Det handler også om å bruke og vurdere digitale verktøy til beregninger, problemløsning, simulering og modellering (Sanne mfl., 2016).

"Det vil si å finne informasjon, analysere, behandle og presentere data med formålstjenlige verktøy, og være kritisk til kilder, analyser og resultat. Videre innebærer det å bli stadig mer oppmerksom på den nytten digitale verktøy har for læring i matematikkfaget"(Sanne mfl., 2016, s. 52).

1.1.2 Fagfornyelsen

I 2016 presenterte regjeringen en Stortingsmelding (nr.28) (Kunnskapsdepartementet, 2016) om behovet for en fornyelse av Kunnskapsløftet. I meldingen stod det blant annet at selv om Kunnskapsløftet hadde bidratt til økt oppmerksomhet og større vekt på elevenes faglige læringsutbytte og grunnleggende ferdigheter, var det likevel fortsatt store utfordringer i grunnopplæringen som ikke er løst. Begrunnelsen var at mange elever hadde et for svakt faglig utbytte av opplæringen, og dette førte til at altfor mange elever ikke fullførte videregående opplæring (Kunnskapsdepartementet, 2016). Det vil si at grunnopplæring har en stor betydning for elevenes fremtid, og det er dermed viktig med stadig vurdering av innholdet i opplæringen slik at det er i tråd med endringene i samfunnet (Kunnskapsdepartementet, 2016; Wangen, 2020).

På denne bakgrunnen anerkjente Regjeringen et behov for videreføring og fornyelse av læreplanverket for Kunnskapsløftet (Kunnskapsdepartementet, 2016). Målet med fornyelsen var å gi en bedre sammenheng mellom de ulike delene av læreplanverket, slik at både skolens brede formål og elevenes faglige læring ville bli bedre ivaretatt (Kunnskapsdepartementet, 2016). I det nye læreplanverket var innsatsen å fremheve de grunnleggende ferdighetene som en del av det faglige innholdet, og en forutsetning for elevenes læring i faget (Kunnskapsdepartementet, 2016).

Med fagfornyelsen ville Kunnskapsdepartementet sikre at fornyelsen fører til at skolefagene videreutvikles slik at de legger bedre til rette for elevenes dybdelæring og grunnleggende kompetanse i fagene; i den forbindelsen understreker departementet viktigheten av konkrete vurderinger av hva læreplanene skal inneholde, av fagpersoner (Kunnskapsdepartementet, 2016).

I 2018 besluttet Kunnskapsdepartementet hva som blir det viktigste i hvert fag, de såkalte kjerneelementene. Grunnen for beslutningen var at lærere og rektorer mente at dagens læreplaner har for mye temaer, og det er vanskelig å gå i dybden på de viktigste fagene grunnet begrenset tid. Dette kunne føre til for mye overfladisk læring i skolen (Kunnskapsdepartementet, 2018b).

Innretningen på mange av fagene ble mot mer praktiske og mindre teoritunge slik at elevene får rom til å gå i dybden på fagene, se sammenhenger mellom fagområder og utvikle evnen til å reflektere og tenke kritisk (Kunnskapsdepartementet, 2018a).

For matematikkfaget ble det introdusert følgende 6 kjerneelementer (Kunnskapsdepartementet, 2018a):

- Utforsking og problemløsing
- Modellering og anvendelser
- Resonnering og argumentasjon
- Representasjon og kommunikasjon
- Abstraksjon og generalisering
- Matematiske kunnskapsområder

Endringer i matematikkfaget er blant annet (Kunnskapsdepartementet, 2018a):

- Elevene skal jobbe mer med metoder og tenkemåter slik at de får større forståelse for faget.
- Tall og tallforståelse er grunnmuren i det elevene skal mestre i løpet av grunnskolen.
- Personlig økonomi, måling og statistikk er viktige områder der tall benyttes i realistiske sammenhenger.
- Programmering og algoritmisk tankegang blir også en del av faget.

Når det gjelder matematikkfaget på videregående skole skal det vektlegges på teoretiske verktøy, problemløsning og resonnering som er viktig for videre studier for teoretisk matematikk, og den praktiske nytten av matematikk for praktisk matematikk. Dette innebærer altså å legge til rette for at elever skal kunne bruke matematikk til å modellere og utforske problemer knyttet til hverdagsliv og samfunn. For de yrkesfaglige matematikkfagene er det utviklet egne kompetansemål der det blir tatt den delen av matematikk som er relevant til yrkesretningen (Utdanningsdirektoratet, 2020).

Sentrale elementer i de nye læreplaner er utforskning og problemløsning. Utforskning legger vekt på at elever leter etter mønster, finner sammenhenger og diskuterer seg fram til en felles forståelse. Strategier og fremgangsmåter er viktigere enn løsninger (Utdanningsdirektoratet, 2020).

Algoritmisk tenkning er en problemløsnings strategi som er tydeliggjort i læreplanen. Det handler om å gi elever gode begreper og effektive verktøy for å løse oppgaver og problemer. Programmering er et kraftig verktøy som kan brukes på en rekke matematiske områder som tidligere var utilgjengelig for elever for eks. i utforskning av store datasett (Utdanningsdirektoratet, 2020).

Kompetansemålene på de første trinnene har søkelys på at elever skal kunne lage og følge regler og trinnvise instruksjoner i lek og spill. Videre skal elever kunne lage og programmere algoritmer ved bruk av variabler, vilkår og løkker. De skal utforske hvordan algoritmer kan lages, testes og forbedres, og de skal bruke programmering til å utforske matematiske egenskaper og sammenhenger (Utdanningsdirektoratet, 2020).

Programmering bidrar med kritisk tenkning og resonering, og stimulerer elvenes kreativitet og fantasi ved å løse problemstillinger. Det vil hjelpe elevene til å skape digitale løsninger ved å omsette en idé til en handling (Sevik mfl., 2018).

1.1.3 Muligheter og utfordringer

Til tross for alle fordelene ved å innlemme programmering i matematikkfaget, vil dette skape utfordringer også, spesielt for matematikklærere. Angående pedagogiske aktiviteter i klasserommet, vil lærere spille en sentral rolle på grunn av det skal være behov for matematikklærere å undervise programmering. Selv om integrering av programmering i matematikkfaget er en politisk beslutning, er en omfattende diskusjon om nødvendige kompetanser for matematikklærere. Mens programmering er innlemmet i læreplaner for matematikkfaget, er det også svært betraktelig diskusjon om å innlemme programmering i utdanningsplan for pre-service (Førtjeneste lærere) og in-service lærere (Lærere i tjenesten) (Forsström & Kaufmann, 2018).

Mens innføring av programmering i læreplaner skal sikre at elevene tilegner seg de nødvendige ferdighetene, må programmering inkluderes også i lærerutdanning, samt å innføre tiltak som støtter lærere i undervisning og programmerings opplæring. Det kan være en utfordrende oppgave å undervise i et programmeringsspråk spesielt for lærere som ikke underviser IKT eller informatikk, og lærere som ikke har hatt tidligere trening i dette området (Balanskat & Engelhardt, 2015).

Jakobsen (2019) undersøker seks argumenter for og mot bruk av programmering i matematikk hvor hun nevner at innføring av programmering i matematikk vil føre til stofftrengsel. “Det er enighet blant informantene[1] om at programmering vil føre til stofftrengsel. Å lære programmering tar tid, og denne tiden vil kunne gå på bekostning av temaer i matematikkfaget. Flere av matematikkfagene i den videregående skolen har også fått redusert temaene i læreplanen for å få plass til programmering. Det trekkes likevel frem at selv om programmering fører til stofftrengsel, kan innføringen også føre til bla. endrede undervisnings metoder og en annen innfallsvinkel til en del av pensum i matematikkfaget”(Jakobsen, 2019, s. 55).

1.2 Lærerkompetanse og relevans

I følge Jakobsen (2019) anses lærerkompetanse, og at lærere blant annet får tid og mulighet til etter- og videreutdanning innen programmering som absolutt viktigst for å lykkes med implementering av programmering som en naturlig del av matematikkfaget i fremtiden. Det er viktig at lærerne også oppfatter programmering som noe relevant i forhold til deres fag (Jakobsen, 2019).

Haraldsrud mener at det vil ta tid at lærerne skaffer seg god kompetanse innen programmering, og man må regne med at det tar noen år med en innkjøringsfase. Han påpeker at det er viktig med kurs som viser gode eksempler på oppgaver som passer ulike trinn slik at lærerne kan ta med seg oppgaver direkte inn i klasserommet som de har allerede jobbet med. Denne typen undervisningen kaller han for undervisnings nære kursing. Lærerne ved hjelp av disse oppgavene kan se hvordan man kan treffe og motivere ulike typer elever i klasserommet (Jakobsen, 2019).

Det er derfor først og fremst viktig med lærerkompetanse for å lykkes ved innføring av programmering i matematikkfaget, og dette innebærer at lærerne som skal undervise programmering får tilrettelagt for kurs og/eller etterutdanning. Det er også viktig at lærerne får mulighet til både å lage og dele gode opplegg (Jakobsen, 2019).

“For lærerne vil programmering for fagets skyld være det som er viktigst, og ikke opplæringen i programmering i seg selv”(Jakobsen, 2019, s. 65).

I forbindelse med behov for å støtte lærere til å anvende programmering i matematikkfaget, skal resultatet av dette prosjektet benyttes som læremateriell av matematikklærerne i tjenesten. Resultatet er en samling av kategoriserte matematiske oppgaver, og algoritmer og programkoder til deres løsninger.

[1] Faglige personer som har deltatt på intervjuet.

1.3 Oppgavens tittel

Den opprinnelige tittelen for oppgaven, «Programmering i STEM-fag: Matematisk modellering algoritmer og programkode som støtter lærere i programmerings undervisning.» er endret til denne nåværende tittelen, «Anvendelse av programmering i matematikk: Matematisk modellering, algoritmer og programkoder som støtter lærere i programmerings undervisning på den norske videregående skolen».

Grunnen for denne endringen var at den opprinnelige tittelen dekket flere fagområder, og begrenset tid for en bacheloroppgave lot ikke å gjennomføre hele oppgaven. Dermed ble det bestemt å begrense oppgaven til kun matematikkfag på videregående skole.

1.4 Problemstilling

Basert på informasjon som ble gitt i forrige delene, er det behov for materialer som støtter lærerne i deres undervisning i matematikkfaget, og som samtidig er i henhold til de nye læreplanene. Innføringen av programmering som en del av matematikkfaget har skapt utfordringer for matematikklærere, og det vil ta tid at lærerne blir gode til å bruke programmering i undervisning av deres fag.

I denne bacheloroppgaven forsøkes å

1 utforske bruk av programmering i matematikkfaget, og

2 utvikle matematiske algoritmer og programkoder som samsvarer de nye læreplaner og støtter matematikklærere i programmerings undervisning på VGS.

Hensikten med oppgaven er å finne ut hvilke matematiske oppgaver egner seg best for å øke dybdeforståelse, engasjement og kreativitet hos elever, og anbefalte fremgangsmåter for implementasjon av disse.

Første delen av problemstillingen handler om å utforske innføring av programmering i matematikk, for eksempel å utforske behovet for innføring av programmering i matematikk og hva programmering skal bidra med i undervisningen av matematikkfaget.

For andre delen skal det lages en samling av matematiske oppgaver som skal løses ved hjelp av programmering. For å iverksette løsningen skal det utvikles algoritmer som viser fremgangsmåten for løsningen.

1.5 Relatert arbeid

I forbindelse med programmering i matematikkfaget på VGS finnes det allerede flere nettsider som tilbyr opplæring både for lærere og elever. Noen eksempler av slike nettsider er:

- **Campus Matte:** Campus Matte er utviklet til fagfornyelse med vekt på dybdelæring og tilpasset opplæring (Campus-Inkrement, udatert).
- **ProFag – realfaglig programmering:** ProFag er etterutdanning for lærere i PROgrammering for FAGenes skyld. Fagene er naturfag, biologi, kjemi, fysikk og matematikk. Tilbudet er spesielt rettet mot realfags lærere med tanke på fagfornyelsen 2020 (UiO, 2021).

- **Kikora:** Kikora tilbyr opplæring i henhold til de nye læreplaner for inklusiv programmering, GeoGebra og CAS for elever på alle trinn (Kikora, udatert).
- **FLIPCLASS:** FLIPCLASS tilbyr videoopplæring i Pythonprogrammering i forbindelse med fagfornyelsen fra høsten 2020 både for elever og lærere (FLIPCLASS, udatert).
- **Programmering i matematikk:** Programmering i matematikk er et etterutdanningskurs for lærere på VGS (Skrindo, udatert).

Det finnes også bøker og hefter som har samme funksjonalitet som disse nettsidene. Eksempler for slike bøker er boka ≪Programmering for matematikklærere≫ av Henning Bueie, og ≪Programmering i skolen≫ av Andreas Drolsum Haraldsrud, Henrik Andersen Sveinsson og Henrik Hillestad Løvold.

Blant disse underviser FLIPCLASS-nettsiden programmering ved å vise først algoritmen og deretter kode oppgaven (ifølge et eksempel video på nettsiden), noe som er nærmest til det som er tenkt å være resultat av dette prosjektet.

1.6 Rapportens struktur

Rapporten inneholder følgende kapitler:

- **Introduksjon og relevans**: I dette kapittelet åpnes temaet til oppgaven, og fordypes problemstillingen; det blir til slutt forklart hvorfor det er behov for gjennomføring av en slik oppgave.
- **Teori**: I dette kapittelet skal vi gå gjennom de sentrale begreper relatert til fagfornyelsen og innføringen av programmering i matematikkfaget som for eksempel dybdelæring, problemløsning, og algoritmisk tenkning.
- **Valg av teknologi og metode**: I dette kapittelet blir Python og dens biblioteker og moduler, og Jupyter Notebook introdusert som de valgte teknologier. Det skal også presenteres DSR-metoden som forskningsmetode ved gjennomføring av oppgaven.
- **Resultater**: I dette kapittelet vil det bli gjennomgått noen av de matematiske oppgaver som er utarbeidet som produkt av bacheloroppgaven. Vi går ikke gjennom alle oppgavene på grunn av stor mengde av resultater, men dokumentet skal legges til rapporten som vedlegg.
- **Diskusjon**: I dette kapittelet skal resultatene bli diskutert og relatert til teoridelen.
- **Konklusjon og videre arbeid**: Dette kapittelet handler om hva som kan bidra til å gjøre det lettere for lærere å anvende programmering i matematikk, og hva som er anbefalt å gjennomføres som videre arbeid til denne oppgaven.

2 Teori

Etter fagfornyelsen er dybdelæring synliggjort i de nye læreplaner. Det finnes flere faktorer som bidrar med dybdelæring, og blant disse faktorene regnes problemløsning som en god metode for at elevene lærer i dybden. Videre finnes det flere problemløsnings strategier og igjen blant disse strategiene regnes algoritmisk tenkning som den viktigste.

For at elevene skal bli gode problemløsere, trenger de å tenke kritisk, logisk og algoritmisk. Programmering egnet seg til å være et godt verktøy for dette formålet. Dessuten har elevene som fremtidige arbeidstakere og samfunnsborgere behov for å mestre utfordringer som de vil møte ved å leve i et stadig mer komplekst samfunn, og programmering regnes som en avgjørende digital ferdighet i denne sammenheng.

I dette kapittelet leser dere blant annet:

- Hva er dybdelæring og hvorfor er det viktig å lære i dybden på skoler?
- Hva innebærer dybdelæring i matematikkfaget?
- Hva er kjerneelementer i matematikkfaget etter fagfornyelsen?
- Hva er problemløsning og hva hjelper problemløsning med i skolesammenheng?
- Hva innebærer begrepet algoritmisk tenkning, og hvordan tenke algoritmisk?
- Hva er programmering, og hvorfor programmering bør innlemmes i skolefag og spesielt i matematikkfaget?

2.1 Dybdelæring

Begrepet dybdelæring ble først brukt for over 40 år siden i forbindelse med en studie gjennomført av Ference Marton og Roger Säljö. De brukte begrepene "surface level processing"og "deep level processing", som kan oversettes til overflatelæring og dybdelæring.

Resultatet av studien viste at studentene som benyttet seg av læringsstrategien overflatelæring, var mer opptatt av å pugge regler og fakta i stedet for å være oppmerksomme om denne i en større sammenheng; mens studentene som lærte seg gjennom dybdelæring, var mer opptatt av å se og forstå sammenhenger. Denne gruppen av studenter viste også en indre motivasjon til å lære og forstå utover god prestasjon i skolesammenheng (Jakobsen, 2019).

Ifølge (Sawyer, 2006) har samfunnet hatt radikale forandringer på grunn av den teknologiske revolusjonen. Disse forandringer har ført til at samfunnet er blitt mer komplekst, og det er dermed behov for ny kunnskap hos samfunnsborgere. I denne forbindelsen blir rollen til utdanningssektoren svært viktig siden elevene er de fremtidige arbeidstakere og samfunnsborgere i det komplekse samfunnet. Dermed blir tanken om at elevene må lære i dybden svært sentralt (Wangen, 2020).

I 2016 løftet regjeringen i Stortingsmeldingen nr. 28 (2015-2016) (Kunnskapsdepartementet, 2016) behovet for en fornyelse av kunnskapsløftet (LK06). Meldingen var basert på arbeidet som undersøkte hvilke grunnleggende kompetanser kreves i fremtidens skole. Med dette arbeidet gjennomførte Ludvigsen-utvalget[2] to utredninger: NOU 2014:7 ≪Elevenes læring i fremtidens skole – Et kunnskapsgrunnlag≫ og NOU 2015:8 ≪Fremtidens skole – Fornyelse av fag og kompetanser≫. I begge utredningene var søkelyset på dybdelæring i skolen noe som ble videreført som et sentralt tema i fagfornyelsen (Wangen, 2020).

[2] "Ludvigsen-utvalget er et utvalg som ble satt sammen for å vurdere grunnopplæringen opp mot hvilke kompetanser som blir viktige i fremtiden"(Wangen, 2020, s. 2).

Ludvigsenutvalget (2015) presenterer følgende definisjonen for dybdelæring:

"Dybdelæring dreier seg om elevenes gradvise utvikling av forståelse av begreper, begrepssystemer, metoder og sammenhenger innenfor et fagområde. Det handler også om å forstå temaer og problemstillinger som går på tvers av fag- eller kunnskapsområder. Dybdelæring innebærer at elevene bruker sin evne til å analysere, løse problemer og reflektere over egen læring til å konstruere en varig forståelse"(Ludvigsenutvalget, 2015).

Utvalget henviser til forskning som viser at dybdelæring har betydning for elevenes utvikling, og er viktig for når de skal fungere godt som arbeidstakere og selvstendige samfunnsborgere i et mer komplekst samfunn (Ludvigsenutvalget, 2014).

I utredningen NOU (2015) påpekes stofftrengsel som utfordring når det skal tilrettelegges for varig læring og progresjon. Det å lære noe i dybden tar tid, og omfattende temaer i et fag, vil gå på bekostning av å arbeide i overflaten med flere av de temaene (Ludvigsenutvalget, 2014).

De mener at kompetanser som kritisk tenkning, kreativitet, metakognisjon, kompetanse i samarbeid og kompleks problemløsning bør settes i fokus for at elevene får en bredere kompetanse enn de får i dag (Wangen, 2020).

I dette prosjektet tas hensyn til dybdelæring som et nøkkel tema, og oppgavene velges slik at de kan bidra med dybdeforsåelse hos elevene. Det har hele veien tatt hensyn til dette for gjennomføring av prosjektet.

2.1.1 Dybdelæring i matematikk

I følge Brekke og Gjone er ordet matematikk ofte knyttet til regning, symboler og formler, heller enn å finne mønster og se etter sammenhenger. Elevene tenker ikke på hva symbolene representerer, og manipulerer dem etter visse regler og formler. Grunnen kan være at den tradisjonelle opplæringen i matematikk som ofte praktiseres i skolene, satser ikke på verken den praktiske, intellektuelle eller sosiale delen av matematikken (Jakobsen, 2019).

Som andre fag har Utdanningsdirektoratet lagt vekt på dybdelæring i de nye læreplaner for matematikkfaget også. For at elevene skal kunne tilegne seg forståelse, trenger de mye tid til å arbeide med fagets kjerneelementer. Utdanningsdirektoratet har derfor lagt til rette for at elever arbeider med færre tema hvert år, og dette skal sikre progresjon og tid til å utvikle forståelse igjennom å utforske sammenhenger i faget (Utdanningsdirektoratet, 2020).

I læreplanene er det også lagt vekt på tema som problemløsning, og at elevene er i stand til å oppdage sammenhenger i, og mellom fagets kunnskapsområder og andre fags kunnskapsområder. Det er ment at å finne slike sammenhenger vil bidra med dybdelæring og forståelse i faget. Et annet tema som er fremhevet i læreplanen for matematikkfaget er å utforske matematikken og kommunisere om den. Læreplanen er bygget opp på den måten at elevene kan knytte matematikken til deres hverdagsliv, og slik de skal være forberedt for å møte utfordringer ved stadig endring i samfunnet og i det fremtidige arbeidslivet. Blant de temaene i læreplanen er algoritmisk tenkning (AT), og programmering også synliggjort med tanke på at disse er viktige problemløsnings strategier som kan utvikle matematisk forståelse (Utdanningsdirektoratet, 2020).

For valg av oppgavene i dette prosjektet er det tatt hensyn til å velge oppgavene slik at de er i samsvar med de nye læreplanene for matematikkfaget; og de temaene (slik som ble nevnt) som er understreket i læreplaner, har vært i fokus.

2.2 Kjerneelementer i matematikk

I matematikkfaget ble seks elementer fastsatt som kjerneelementer. "De fem første kjerneelementene beskriver arbeidsmåter, metoder og tenkemåter i matematikk. Det sjette kjerneelementet beskriver de sentrale kunnskapsområdene i matematikk. Elevene skal møte det sjette kjerneelementet gjennom de fem første kjerneelementene"(Kunnskapsdepartementet, 2018b, s. 15):

- Utforskning og problemløsning:
  - Utforsking: "Utforsking handler om at elevene leter etter mønstre og finner sammenhenger. Elevene skal legge mer vekt på strategiene og framgangsmåtene enn på løsningene" (Kunnskapsdepartementet, 2018b, s.15).
  - Problemløsning: "Problemløsing handler om at elevene utvikler en løsningsmetode på et problem de ikke kjenner fra før"(Kunnskapsdepartementet, 2018b, s. 15). I denne sammenheng er algoritmisk tenkning nevnt som en viktig metode for å utvikle strategier og fremgangsmåter, og mer presist innebærer å bryte ned et problem i flere delproblemer som kan løses systematisk (Kunnskapsdepartementet, 2018b).
- Modellering og anvendelser: Modellering og anvendelser innebærer at elevene oppfatter og ser bruk av matematikken i dagligliv, samfunnsliv, vitenskap og teknologi. Elevene skal altså være i stand til å hente en problemstilling fra virkeligheten, omformulere den til en matematisk modell, og tolke modellen i forhold til den opprinnelige situasjonen. Det skal også gi elevene innsikten i hvordan modeller kan anvendes i nye situasjoner, og muligheten til å kunne tenke kritisk (Kunnskapsdepartementet, 2018b).
- Resonnering og argumentasjon: Resonnering og argumentasjon handler om at elevene skal forstå matematiske regler, og følge og vurdere matematiske resonnementer. Elevene må lære å utforme egne resonnementer både for å løse matematiske problemer, og for å argumentere for framgangsmåter og løsninger (Kunnskapsdepartementet, 2018b).
- Representasjon og kommunikasjon: "Elevene må få mulighet til å bruke matematiske begreper i ulike sammenhenger gjennom egne erfaringer og matematiske samtaler. Elevene må kunne forklare valgt fremgangsmåte og kunne begrunne svarene sine. Det innebærer også å kunne oversette mellom det matematiske symbolspråket og dagligspråket og veksle mellom ulike representasjonsformer"(Kunnskapsdepartementet, 2018b, s. 15).
- Abstraksjon og generalisering: "Forståelsen for generelle matematiske problemstillinger utgår fra kunnskaper og ferdigheter. Elevene skal forstå representasjoner og fremgangsmåter av økende abstraksjons grad. Elevene bør derfor oppdage sammenhengene og strukturene selv og ikke blir presentert for en ferdig løsning. Dette foregår gjennom å utforske med tall, utregninger og figurer for å finne sammenhenger og deretter å formalisere ved bruk av algebra og hensiktsmessige representasjoner"(Kunnskapsdepartementet, 2018b, s. 15).
- Matematiske kunnskapsområder

De to første punktene er viktige og sentrale temaer i dette prosjektet, og det vil spesielt legges vekt på dem i oppgavene som blir utviklet.

2.3 Problemløsning

"problemløsning er prosessen med å bevege seg mot et mål når veien til målet er usikker"(Martinez, 1998, s. 605).

Problemløsning er en av de kjerneelementene for matematikkfaget (Kunnskapsdepartementet, 2018b), og beregnes som et sentralt element for grunnopplæring i den norske skolen. Pólya (1957)

i boka «How to solve it» presenterer en struktur for problemløsning som er bygd opp av fire hovedpunkter (Jakobsen, 2019):

1. Å forstå problemet
2. Å legge en plan
3. Å gjennomføre planen
4. Å se tilbake

Ved hjelp av problemløsnings oppgaver lærer elevene matematikk, modeller, situasjoner eller kontekster. Problemløsning som en av kjerneelementene i matematikkfaget, og som en viktig kompetanse er innført i de nye læreplaner for matematikkfaget (Jakobsen, 2019). Torkildsen (2017) mener at problemløsnings aktiviteter bidrar til økt forståelse og dybdelæring hos elvene (Torkildsen, 2017). Arbeid med problemløsning gjør elvene i stand til å utvikle en helhetlig matematisk kompetanse. Dette gir etter hvert elevene mulighet til å vurdere de forskjellige strategier og finne den riktige strategien. Metoden vil også hjelpe lærerne til å få innsikt i elevenes tanker og kompetanse (Jakobsen, 2019).

Problemløsning er en avgjørende faktor for at elevene skal lære i dybden. Det å løse problemer vil stimulere elevenes kreativitet, og vil hjelpe dem til å bli gode problemløsere etterhvert. Derfor handler oppgavene som er utviklet i dette prosjektet for det meste om dette temaet. De fleste av oppgavene krever at elevene først må finne en strategi for å kunne løse oppgaven deretter.

2.3.1 Problemløsning ved hjelp av datamaskiner

I dataverdenen er problemløsning en prosess for å identifisere et problem, utvikle en algoritme for det identifiserte problemet, og til slutt å implementere algoritme for å utvikle et program (NCERT, 2020).

Det er enkelt å finne løsning for et enkelt problem, men for komplekse problemer trenger man å tenke algoritmisk. For å løse problemet på en strukturert måte, bør det følges følgende trinn (NCERT, 2020):

1 Analysere problemet

2 Utvikle en algoritme

3 Implementere algoritmen (Programmere)

4 Teste og feilsøke

Sammenligning av denne metoden med Pólya (1957) sin problemløsnings strategi viser at de følger én og samme struktur. Derfor kan det sies for å løse et problem krever å tenke på en systematisk måte uavhengig av fagområdet.

For å løse oppgavene i dette prosjektet, brukes denne strategien. Enhver oppgave består av en del som analyserer oppgaven og finner løsningen, deretter utvikles en algoritme til løsningen, og videre tas i bruk denne algoritmen for å programmere løsningen. Programmet testes ved å benytte et klietprogram, og få output av programmet.

2.4 Algoritmisk tenkning

Futschek (2006) gir følgende definisjonen for en algoritme:

"En algoritme er en metode for å løse et problem som består av nøyaktig definerte instruksjoner"(Futschek, 2006, s. 160).

Han beskriver algoritmisk tenkning som en pool av evner til å

- analysere gitte problemer.
- spesifisere et problem nøyaktig.
- finne de grunnleggende handlingene som er tilstrekkelig for det gitte problemet.
- konstruere en riktig algoritme for et gitt problem ved hjelp av de grunnleggende handlinger.
- tenke på alle mulige spesielle og normale tilfeller av et problem.
- forbedre effektiviteten til en algoritme.

Sevik mfl. (2018) mener algoritmisk tenkning er en problemløsnings prosess hvor man tenker som en informatiker når man skal løse et problem. Den innebærer:

- Å bryte ned store og komplekse problemer til mindre og mer håndterbare delproblemer.
- Å organisere og analysere data på en logisk måte, og å løse store og komplekse problemer ved å lage algoritmer.
- Å lage abstraksjoner og modeller av et virkelig problem og generalisere løsninger slik de kan brukes i liknende tilfeller. Denne arbeidsmåten er sentralt i programvareutvikling, men kan benyttes som metode i andre sammenhenger også.

I matematikk er logisk tenkning om algoritmer, og problemløsning de viktigste lærings utbyttene. Algoritmisk tenkning gir elevene mulighet til å jobbe med algoritmer, og betraktes som en systematisk beskrivelse av en spesifikk tilnærmet løsning (Kaufmann & Stenseth, 2020).

I programmerings verdenen er også algoritmisk tenkning en svært sentral metode (Haraldsrud mfl., 2020). Det handler om å kunne utvikle programmer for å utføre algoritmer (Kaufmann & Stenseth, 2020). For å løse en problemstilling ved hjelp av algoritmisk tenkning er det viktig å sette søkelys på problemstillingen gjennom noen sentrale strategier som systematisering, analyse, gruppering, abstraksjon og evaluering. Når man vil løse et problem ved hjelp av algoritmisk tenkning, er først og fremst viktig å gjenkjenne mønsteret, med andre ord å gjenkjenne strukturer og likheter mellom ulike prosesser og rutiner (Haraldsrud mfl., 2020).

Algoritmisk tenkning er en god måte for at elevene skal bli gode problemløsere. Haraldsrud mfl. (2020) forslår følgende mulige måter for å jobbe med algoritmisk tenkning (Haraldsrud mfl., 2020):

- Vise ulike løsnings strategier for et problem, og la elevene diskutere dem.
- La elevene selv komme fram til mulige løsningsforslag for et problem.
- Gi elevene gjerne oppgaver med små hint underveis i prosessen til problemløsningen, og diskutere hva de kommer fram.
- La elevene fylle ufullstendige programmer og diskutere mulige løsnings strategier.
- La elever skissere programmer før de lager dem.

- La elevene gjerne bruke pseudokode.
- Bruke spill og rollespill med rutiner og mønster som elver skal finne ut av eller manipulere.
- Drøfte ulike formuleringer av hverdagslige algoritmer som hjernen vår behandler daglig, slik som ansikts- og stemmegjenkjenning, hverdagsrutiner og forflytning.

Allikevel finnes det ikke en standard måte å lære algoritmisk tenkning på. Futschek (2006) mener å svare på spørsmålet «Hvordan lære algoritmisk tenking?», er like vanskelig å svare på «Hvordan lære kreativitet?» Et mulig svar kan være å løse mange (løselige) problemer uavhengig av programmeringsspråk. Han mener videre at nybegynnere spesielt har problemer med å forstå de underliggende elementer i et programmeringsspråk riktig. Det er derfor viktig at språket som beskriver algoritmen er høyt nivå og problemorientert, f.eks. pseudokode oppfyller disse kriteriene (Futschek, 2006).

Bueie (2019) i boka «Programmering for matematikklærere» har et godt eksempel på hvordan tenke algoritmisk for å finne kvadratroten av 12. Han skriver, én kan tenke seg at elever uten tilgang til andre hjelpemiddel vil forsøke å gjette seg fram til svaret ved hjelp av multiplikasjon. Det er en god måte å tenke på så lenge løsningen er et heltall. Vi kan overføre tenkemåten til en algoritmisk beskrivelse hvor en prøver ser fram verdi for verdi til én finner riktig svar:

$0 \times 0 \neq 121$
$1 \times 1 \neq 121$
$2 \times 2 \neq 121$
...
$10 \times 10 \neq 121$
$11 \times 11 = 121$

Når vi har funnet mønsteret til løsningen, kan vi ved hjelp av en *while*-løkke lage et program som gjetter løsningen (Bueie, 2019).

De fleste av oppgavene som er utviklet i dette prosjektet krever at elevene tenker algoritmisk, og finner et mønster for å løse oppgaven. Dette gjelder spesielt oppgavene som har fokus på problemløsning, og trengs for å tenke på en logisk måte for å finne løsning til oppgaven.

Etter at løsningene er oppdaget, utvikles en algoritme i form av pseudokode for å programmere dem, siden (som ble nevnt) pseudokode er en god metode for representasjon av algoritmer for nybegynnere.

2.5 Algoritme

Algoritme er en fullstendig og nøyaktig beskrivelse som viser fremgangsmåten for å løse et problem. Den innebærer også å kunne bryte ned et problem i delproblemer som kan løses systematisk, og angir skrittene og rekkefølgen til løsningen av problemet ved ord, matematiske symboler, og/eller skjematisk fremstilling av arbeidsgangen (Jakobsen, 2019).

Før man vil begynne med å skrive et program, er det viktig å lage en løsning. Denne løsningen kan representeres i naturlig språk, og kalles for en algoritme. En algoritme kan sees på som en oppskrift; dersom oppskriften er velskrevet med klare definerte skritt, kan brukeren ende opp med å forberede retten. En god algoritme er nøyaktig, unik, slutter etter et visst antall trinn, mottar input, og gir output. Dette betyr at hver algoritme bør gi mulighet for å ta imot data fra brukeren, behandle dataene som skal utføres for å gi ønsket resultat, og gi output (NCERT, 2020).

Algoritmisk tenkning vil hjelpe elevene til å analysere problemet, og identifisere de logiske skrittene som må følges for å nå en løsning. Det finnes to vanlige metoder for å representere en algoritme på (NCERT, 2020):

- Flytdiagram
- Pseudokode

Ved bruk av enhver av disse metodene bør det tas hensyn til at algoritmen viser logikken til løsningen (og ikke alle detaljer ved implementering), og at algoritmen viser tydelig kontrollflyten under utførelsen av programmet (NCERT, 2020).

Algoritmer er svært sentralt i dette prosjektet; oppgaven i prosjektet er å utvikle algoritmer for matematiske oppgaver. For representasjon av disse algoritmer brukes noen ganger flytdiagram, men ofte skrittvise pseudokoder.

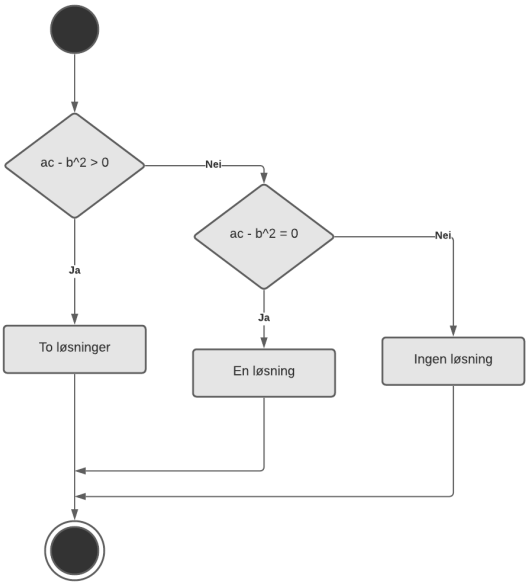
2.6 Flytdiagram

Et flytdiagram er en visuell fremstilling av en algoritme i form av et diagram som består av standardiserte symboler for eksempel bokser, diamanter og andre geometriske former som er koblet sammen med piler hvor hver form representerer en skritt av løsnings prosessen (NCERT, 2020).

Flytdiagram kan altså defineres som en trinnvis visualisering av sekvenser, valg, eller repetisjon av instruksjoner til å løse en oppgave. Figur 2.1 viser et eksempel flytdiagram for å løse andre-gradslikninger ved hjelp av abc-formelen.

Sterneckert (2003) mener det finnes flere typer av flytdiagrammer, som for eksempel for analytikere, designere, ledere, eller programmerere som hver av disse vil legge detaljer i flytdiagrammer basert på sin forståelse:

- Dokument-flytdiagram har som mål å vise eksisterende kontroller over dokumentflyt gjennom komponenter av et system.
- Data-flytdiagram er et diagram som har som mål å vise de kontroller som styrer dataflyt i et system.
- System-flytdiagram er en illustrasjon som viser kontroller plassert på fysisk eller ressursnivå.
- Program-flytdiagram er et diagram som viser kontroller plassert internt i et program i et system.



Figur 2.1: Eksempel flytdiagram: Diagrammet viser antall mulige løsninger for andregradsligninger.

Source: (Bueie, 2019)

2.7 Pseudokode

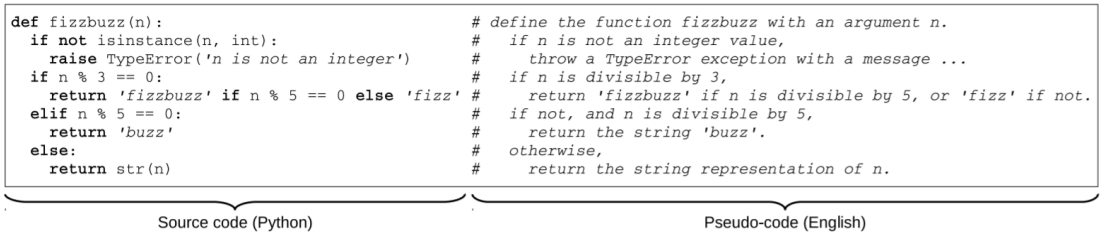
"Pseudokode er en algoritmisk metode som bruker naturlig språk for å gi in-depth, cross-platform kode beskrivelse"(Alhefdhi mfl., 2018).

Ordet «pseudo» betyr «ikke ekte», så «pseudokode» betyr «ikke-ekte kode». Pseudokoding er en metode for å representere en algoritme på. Pseudokoding er en nyttig metode for å beskrive et program i detalj, og på et uformelt språk som kan forstås enkelt av mennesker (NCERT, 2020).

Det finnes flere fordeler med pseudokoding blant annet at pseudokoder programmeres språkuavhengig noe som betyr at én og samme pseudokode kan oversettes til muligens hvilket som helst programmeringsspråk. Pseudokoder beskriver nøyaktig hva som koden gjør trinnvis, ved å bruke naturlig språk og matematiske uttrykk som forstås som regel av de fleste. Pseudokoder forklarer hva hvert kodesnutt betyr, noe som gjør det enklere for nybegynnere å forstå et program i et ukjent programmeringsspråk (Alhefdhi mfl., 2018), med andre ord hjelper pseudokoder å forstå fra bunn til topp av et program ("bottom-up comprehension"(M.-A. Storey, 2006)) (Oda mfl., 2015).

Pseudokoding er også en god problemløsnings strategi i skolesammenheng; elevene ved å utvikle først en pseudokode-representasjon av en løsning på et problem, og deretter å skrive koden utfra den pseudokoden, kan forbedre sine problemløsnings ferdigheter (Olsen, 2005).

I dette prosjektet brukes pseudokoding for å vise algoritmen som løser en matematisk oppgave. Figur 2.2 viser hvordan et kodesnutt i Python kan representeres som pseudokode. Merk at strukturen er den samme i begge tilfeller, men pseudokode er skrevet på et mer forståelig språk.



Figur 2.2: Konvertering av Pythonkode til pseudokode.

Source: (Oda mfl., 2015)

2.8 Programmering

"Programmering går ut på å sette opp en serie instruksjoner som styrer maskinen og som avgjør hvordan den skal reagere på inndata, inntastinger, musebevegelser og annet"(Rossen, 2019).

Et program er koden som skrives av programmerer etter en spesifikk syntaks består av instruksjoner som viser hva programmet skal utføre. Noen av programmeringsspråkene er lavnivå, og noen andre regnes som høynivå språk. Eksempel for lavnivå programmeringsspråk er maskinkode der det finnes få abstraksjoner mellom det som programmerer skriver og instruksjoner som leses av datamaskinen. I et høynivå programmeringsspråk som C++, derimot skrives programmet slik at det enkelt kan forstås av mennesker (Rossen, 2019).

Programmering er ikke bare koding, og den kan sees som en prosess som innebærer aktiviteter også utenfor datamaskin. Når man vil løse et problem ved hjelp av programmering, må først problemet analyseres og deles opp i håndterbare deler. For å løse enkelte problemer skjer denne fasen i hodet, men for større og mer komplekse problemer vil man ha behov for å for eksempel skissere problemet, og å bryte ned problemet i mindre delproblemer. Denne organisasjonen og analyseringen av komplekse problemer kaller Haraldsrud mfl. (2020) for algoritmisk tenkning (Haraldsrud mfl., 2020).

I de nye læreplaner er programmering innført i matematikkfaget med tanke på at programmering utfordrer elevene med forskjellige problemstillinger hvor det er sjelden en løsning på hvordan utfordringen løses, og elevene er nødt til å bruke relevante strategier for å komme i mål, og løse utfordringene de møter mens de jobber med programmering (Sanne mfl., 2016).

Her kan betraktes et svært tett forhold mellom programmering og matematikk gjennom problemløsning siden algoritmisk tenkning[3] kan betraktes som et sett av problemløsnings metoder består av problemer og deres løsninger på den måten som kan utføres av datamaskinen (Kaufmann & Stenseth, 2020).

Programmering brukes som et verktøy for å løse problemer; i dette prosjektet brukes programmering for å løse matematiske problemer.

2.8.1 Programmering i matematikk

Programmering i matematikkfaget ble først introdusert av Seymour Papert i 1980 da han ga ut boka «Mindstorms: children, computers, and powerful ideas». I boka beskriver han hvordan programmeringsspråket LOGO skaper en mikroverden for barna der de kan lære matematikk (Bueie, 2019).

[3] I den originale teksten har forfattere brukt ordet Computational Thinking (CT) som på norsk er oversatt som Algoritmisk Tenkning (AT).

I Norge fikk først programmering noe plass i 1987 da datalære ble knyttet til matematikkfaget i mønsterplanen for grunnskolen (M87) ved å ta utgangspunkt i algoritmebegrepet (Bueie, 2019).

På 1990-tallet var færre skoler som underviste programmering over hele verden. Kafai og Burke (2013) ser grunnen for denne tilbakegangen vanskeligheter ved å innpasse og koble fagstoffet til læringsmål, utilstrekkelig antall kvalifiserte lærere, begrenset tilgang til teknologi, og uklart formål med å undervise programmering på skoler (Kaufmann & Stenseth, 2020).

I de siste årene, har programmering begynt å komme inn igjen på skoler. Elevene er kreative, og bruker teknologi for samhandling, spill, og i en rekke applikasjoner. Til tross for de bruker teknologi hele tiden og overalt, har de ikke tilstrekkelig forståelse for, for eksempel hvorfor å klikke på musen produserer handling på skjermen. Samfunnet i dag forandrer seg stadig raskere, og teknologisk utvikling fører til sosiale utfordringer. Det er dermed behov for en dypere forståelse av teknologi hos elevene som fremtidige samfunnsborgere (Kafai & Burke, 2013).

Flere studier har undersøkt bruk av programmering i matematikk på skoler, og flere av disse studiene hevder at programmering kan føre til bedre matematiske resultater. Noen andre studier indikerer at programmering kan motivere elevene til å studere matematikk. Programmering kan også bidra til at elevene forbedrer deres ferdigheter i problemløsning i matematikk (Kaufmann & Stenseth, 2020).

Siden programmering ofte har vært knyttet til matematisk tenkning, har i flere land vært tendens til å inkludere programmering for å utvikle algoritmisk tenkning (Grover & Pea, 2013). Integrering av programmering i matematikkfaget, gir muligheten til å kombinere fagspesifikke ferdigheter med ferdigheter i programmering. Dette gjør elevene i stand til å knytte teoretiske matematiske begreper opp mot en praktisk kontekst, noe som gir forståelse for kunnskap i en større sammenheng hos elevene (Sevik mfl., 2018). Programmering har potensial til å påvirke elvenes holdning i matematikk, og gir dem mulighet til å koble matematikk til det virkelige livet på en ny måte (Forsström & Kaufmann, 2018). Dette potensialet er viktig i matematikkfaget som oftest oppleves som et ganske isolert skolefag (Lambic, 2011).

Ved integrering av programmering i matematikk, vil formen for undervisning i klasserommet endres. Programmerings aktiviteter tilsvarer ikke de tradisjonelle læringssituasjoner i klasserommet. Å jobbe med programmerings oppgaver krever ofte samarbeid, og læreren spiller en annen rolle enn den vanlige rollen (Forsström & Kaufmann, 2018). Vanlige matematikklærere er nødt til å undervise i programmering, og dette krever at matematikklærere har de nødvendige kompetanser. Samtidig som programmering er innlemmet i matematikkplanen, er det en svært betydningsfull diskusjon også om å innlemme programmering i pre-service og in-service lærerutdanningen (Forsström & Kaufmann, 2018).

"Hvilken teknologi som vil være i bruk, og hvilket grensesnitt som skal brukes i klasserom i årene som kommer, er vanskelig å spå. En fellesnevner ser ut til å være at den som skal undervise i programmering i faget matematikk, bør ha god kunnskap om programmering"(Bueie, 2019).

3 Valg av teknologi og metode

I dette kapittelet leser dere om valg av teknologi og metode. Teknologidelen inneholder oversikt over de valgte teknologier og grunnen for valg av disse. Metodedelen handler om forskningsmetoden DSR, og hvordan denne metoden er benyttet i prosjektet.

3.1 Teknologi

For gjennomføring av dette prosjektet ble det valgt å programmere i Python ved hjelp av programmet Jupyter Notebook. Grunnen for valget er at Python har mange styrker, og det forventes at dette blir det naturlige valget i matematikkundervisnings sammenheng i årene fremover (Bueie, 2019). Språket er lett å lære og har moduler og biblioteker for ethvert formål. Dessuten er Python fritt tilgjengelig, og kan kjøres med relativt liten maskinkraft. Python-tolker installeres på flere typer operativsystemer, og slik kan Pythonprogram kjøres på et stort utvalg av systemer (Bueie, 2019). Språket er egnet for å løse matematiske problemer, og derfor anses som det mest hensiktsmessige programmeringsspråket i dette prosjektet.

3.1.1 Python

Python er et høynivå, objektorientert programmeringsspråk som ble lansert på starten av 90-tallet av Guido van Rossum. Guido van Rossum begynte å jobbe med Python som en etterfølger for ABC-programmeringsspråket, og ga den først ut i 1991 som Python 0.9.0. Neste versjonen av Python (Python 2.0) ble utgitt i 2000 med mange viktige nye funksjoner, inkludert en «cycle-detecting garbage collector» og støtte for Unicode, og avviklet med versjon 2.7.18 i 2020. Python 3.0 ble utgitt i 2008 og var en større revisjon av språket som ikke er helt bakover kompatibelt, og mye av Python 2-koder kjører ikke umodifisert på Python 3 (Van Rossum mfl., 2007).

Pythons designfilosofi legger vekt på lesbarhet, og syntaksen gir programmerere mulighet til å uttrykke konsepter på færre kodelinjer enn i andre programmeringsspråk som C++ og Java. Konstruksjoner i Python er fleksible til å muliggjør klare programmer både i liten og store skala (Tulchak & Marchuk, 2016). Noen typiske bruksområder for Python er automatisering, dataintegrering og dataanalyse (Dvergsdal, 2019).

Selv om Python har mye i felles med andre programmeringsspråk, finnes det noen språklige kjennetegn som skiller Python fra de andre språkene (Dvergsdal, 2019):

- Definering av kodeblokker ved hjelp av innrykk: i Python markeres hver blokk ved hjelp av innrykk mens andre programmeringsspråk har egne tegn eller reserverte ord for markering av start og slutt for blokker. For eksempel i Java markeres hver blokk av kode med «{» i starten og «}» i slutten.
- Alle verdier er objekter: I Python er alle verdier immutable objekter ved definering. Dette særtrekket gir mulighet for mer fleksibilitet samtidig som språket er ryddig og lett å utvides. Det gir også stor frihetsgrad til hvordan verdiene lagres.

3.1.2 Python: biblioteker og moduler

Matplotlib Matplotlib er et Python-bibliotek for å lage statistiske, animerte og interaktive visualiseringer. Biblioteket ble opprinnelig skrevet av John D. Hunter, og siden den tiden (2002) har vært aktivt under utvikling. I 2012 ble Michael Droettboom nominert som matplotlibs hoved-utvikler rett før John Hunters død. I dag benyttes biblioteket av tusenvis av forskere over hele verden (Hunter mfl., 2021b).

"Matplotlib ble født i det vitenskapelige databehandlings området, hvor gnuplot og MATLAB ble brukt (og brukes fortsatt) mye"(Tosi, 2009, s. 8).

Matplotlib ble modellert på MATLAB, fordi graftegning var noe som MATLAB klarte seg veldig bra. Høy grad av kompatibilitet mellom MATLAB og Matplotlib førte til at mange MATLAB-brukere tok i bruk Matplotlib. Biblioteket er designet med tanke på å bruke Python som språk, men å ha MATLAB-funksjonaliteten, og i tillegg har denne fordelen å være gratis og open-source (Tosi, 2009).

- Pyplot: matplotlib.pyplot er en samling av funksjoner som får matplotlib til å fungere som MATLAB. Modulen inneholder funksjonalitet for plotting og visualisering; *plot()*-funksjonen er et eksempel blant funksjoner for plotting fra denne modulen (Hunter mfl., 2021f).
- Pylab: Pylab er en modul som inkluderer matplotlib.pyplot, numpy, numpy.fft, numpy.linalg, numpy.random, og noen andre tilleggs funksjoner. Formålet med å opprette en slik modul var å kunne importere alle MATLAB-lignende funksjoner under et globalt navnområde. Men denne typen importering regnes som en dårlig stil i dag siden den kan føre til ikke-forventet oppførsel. Dermed frarådes sterkt bruk av Pylab, og anbefales i stedet å bruke matplotlib.pyplot (Hunter mfl., 2021a).

NumPy NumPy er et open-source prosjekt rettet mot å bli brukt for numerisk databehandling i Python. Biblioteket ble opprettet først i 2005, og er bygd på bibliotekene Numeric og Numarray. NumPy regnes som et sterkt vitenskapelig bibliotek for databehandling hos nesten alle forskere som arbeider i Python. Denne beregnings kraften er overført fra programmeringsspråkene C og Fortran til Python som er et mye lettere språk å lære og bruke (NumPy, udatert).

Biblioteket har oppsummert følgende streke sider:

- Kraftige n-dimensjonale arrayer
- Numeriske regnskapsverktøy
- Interoperabilitet
- Ytelse
- Lett å bruke
- Open-source

SymPy SymPy er et Python-bibliotek for symbolsk matematikk. Biblioteket er ment å være et full-featured Computer Algebra System (CAS), og samtidig holde koden så enkelt som mulig for å være forståelig og utvidbar (SymPy-Development-Team, udatert).

SciPy SciPy er et open-source økosystem for vitenskapelig databehandling (Scientific Computing) i Python som bygger på kjerne av følgende pakker (SciPy-Developers, udatert):

- NumPy: Den grunnleggende pakken for numeriske beregninger.
- SciPy: En samling av numeriske algoritmer og domenespesifikke verktøykasser, inkludert signalbehandling, optimalisering, statistikk og flere andre.
- Matplotlib: En moden og populær pakke for plotting.
- IPython: IPython gir en rik arkitektur for interaktiv databehandling med (IPython-Development-Team, udatert):
  - Et kraftig interaktivt skall.
  - En kjerne for Jupyter.
  - Støtte for interaktiv datavisualisering og bruk av GUI-verktøysett.
  - Fleksible, innebygde tolker som kan lastes inn i dine egne prosjekter.
  - Brukervennlige verktøy med høy ytelse for parallell databehandling.
- SymPy: En pakke for symbolsk matematikk og Computer Algebra.
- Pandas: pandas er et raskt, kraftig, fleksibelt og brukervennlig verktøy for open-source data-analyse og manipulering, bygget på Python (pandas-Development-Team, udatert).

Astropy Astropy er et open-source prosjekt som har som mål å utvikle en felles kjernepakke for astronomi i Python, og å fostre brukervennlighet, interoperabilitet og samarbeid mellom astronomi-pakker (Astropy-Team, udatert).

Math Math er en modul i Python som inkluderer matematiske funksjoner definert av C-standarden. Funksjoner i Math-modulen er ikke definert for komplekse tall, og for beregninger med komplekse tall, bør det benyttes cmath-modulen som har funksjoner med samme navn som i Math, men som støtter komplekse tall (Python-Software-Foundation, 2021b).

Her nevnes noen kjente funksjoner fra denne modulen (Python-Software-Foundation, 2021b):

- math.exp(x): Funksjonen tar x som parameter og returnerer e med potens x, der $e = 2.718281\ldots$ er basen til naturlige logaritmer. Den returnerte verdien av *exp()*-funksjonen er mer nøyaktig enn math.e**x eller pow (math.e, x).
- math.sqrt(x): Funksjonen tar x som parameter og returnerer kvadratroten til x.
- math.pi: Den matematiske konstanten $\pi = 3.141592\ldots$, med tilgjengelig nøyaktighet.
- math.sin(x): Funksjonen tar x som parameter og returnerer sinus til x i radianer.
- math.log (x [, base]): Hvis det mates x som parameter, returnerer funksjonen den naturlige logaritmen til x (med base e). Hvis det gis to parametere (x og basen), returnerer funksjonen logaritmen til x til gitt base, beregnet som $\log(x)/\log(base)$.

Random Random er en built-in modul i Python, og brukes for å frembringe pseudo-tilfeldige variabler. For eksempel å få et tilfeldig tall, velge et tilfeldig element fra en liste, blande tilfeldig elementer i en liste osv. (Python-Software-Foundation, 2021c).

Time Time-modulen har forskjellige funksjoner som er relatert til tid (Python-Software-Foundation, 2021d).

3.1.3 Jupyter notebook

Jupyter er et open-source prosjekt som ble startet i 2014 i fortsettelse av IPython-prosjektet for å støtte interaktive datavitenskap og vitenskapelig databehandling på tvers av alle programmeringsspråk (Project-Jupyter, 2021).

Jupyter Notebook (tidligere IPython Notebooks) er en open-source webapplikasjon som gir mulighet for å lage og dele dokumenter som inkluderer live-code, likninger, visualiseringer og narrative tekster. Bruksområder for Jupyter Notebook er for eksempel datarengjøring (data cleaning på engelsk) og transformasjon, numerisk simulering, statistisk modellering, datavisualisering, maskinlæring og flere andre (Project-Jupyter, 2021).

Jupyter Notebook er tilgjengelig via en nettleser, og det gir mulighet for å bruke samme grensesnitt som kjører lokalt, som en stasjonær applikasjon, eller på en ekstern server. I sistnevnte tilfelle er den eneste programvaren brukeren trenger lokalt, en nettleser; så for eksempel kan en lærer sette opp programvaren på en server og enkelt gi elever tilgang (Kluyver mfl., 2016).

NoteBook-fil er et enkelt dokument på JSON-format, med utvidelsen ≪.ipynb≫ (Kluyver mfl., 2016). Dokumentet inneholder en liste av celler som kan inneholde kode, tekst (ved bruk av Markdown), matematikk, plott osv., og kan også konverteres til for eksempel HTML, LaTeX, PDF, Python osv. .

3.2 Metode

Metoder definerer prosesser for å finne ut hvordan et problem kan løses. Metodene avhengig av hva slags problem som skal løses varierer. De er enten formelle, uformelle, eller en kombinasjon av begge. Metodene hjelper med å finne ut hvor gjennomførbar en artefakt er, og gir mulighet til konkrete vurderinger av artefakten i forhold til dens formål. De hjelper forskere å lære om verden og gjør dem i stand til å vurdere artefakten i den virkelige verdenen (Hevner mfl., 2004).

Dette prosjektet er gjennomført ved hjelp av forskningsmetoden ≪Design Science Research≫ som blir forklart i neste delkapittelet.

3.2.1 Design Science Research

“Design-Science er fundamentalt et problemløsnings paradigme som har sin opprinnelse i ingeniørfag og viten til en artefakt (artifact på engelsk). Metoden søker å skape innovasjoner definert av ideer, praksiser, tekniske evner, og produkter som kan være effektivt og effektivt oppnådd, gjennom analyse, design, implementering, styring og bruk av informasjons systemer (IS)”(Hevner mfl., 2004, s. 76).

Opprettelsen av artefakter (som produseres ved bruk av DSR-metoden) er avhengig av eksisterende kjerneteorier (kernel theories på engelsk) som brukes, testes, modifiseres og utvides gjennom opplevelse, kreativitet, intuisjon og problemløsnings evner til forsker (Hevner mfl., 2004).

I IS-litteraturen slik Hevner mfl. (2004) nevner, er designprosessen viktig, og Benbasat og Zmud (1999) hevder relevansen av IS-forskning er direkte relatert til dens anvendbarhet i design (Hevner mfl., 2004).

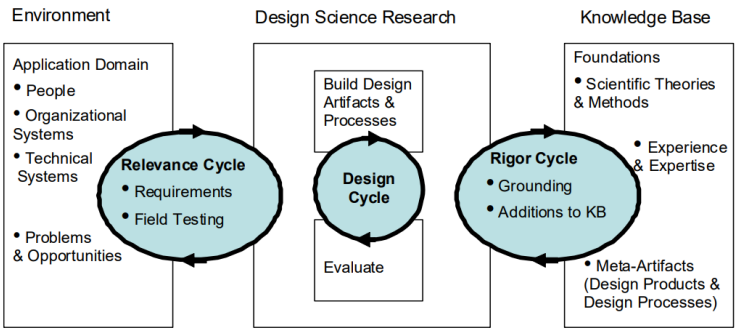
“Den resulterende IT-artefakten utvider grensen for menneskelig problemløsnings- og organisatoriske evner ved å tilby intellektuelle verktøy i tillegg til beregnings verktøy”(Hevner mfl., 2004, s. 76).

I designprosessen produseres et innovativt produkt (artefakt) gjennom en sekvens av ekspertaktiviteter. Det er viktig at forskeren evaluerer artefakten. Tilbakemeldinger fra evalueringen vil gi en bedre forståelse av problemet, og basert på disse kan forskeren forbedre kvaliteten av produktet og designprosessen. Build-and-evaluate-loopen gjentas vanligvis flere ganger før den endelige artefakten er ferdigstilt (Hevner mfl., 2004).

Hevner mfl. (2004) definerer 7 retningslinjer for gjennomføring av en design-science forskning med tanke på å hjelpe forskere, anmeldere, redaktører og lesere for å forstå kravene til en effektiv design-science forskning (Hevner mfl., 2004):

- Etablering av en innovativ og målrettet artefakt.
- Spesifisering av problemdomene.
- Evaluering av artefakten slik som den er nyttig.
- Artefakten er nyskapende, og løser problemet på en effektiv måte.
- Artefakten må være grundig definert, formelt representert, sammenhengende, og internt konsistent.
- Finne en effektiv løsning mens den endelige artefakten tilfredsstiller lover i miljøet problemet er reist.
- Resultatet kommuniserer effektivt både tekniskorientert og ledelsesorientert.

I artikkelen «A Three Cycle View of Design Science Research» fokuserer Hevner (2007) på tre av disse retningslinjer som "tre iboende forsknings sykluser"(Hevner, 2007). Figur 3.1.



Figur 3.1: Design Science Research Cycles

Source: (Hevner, 2007)

Relevance Cycle Relevanssyklusen bygger bro mellom det kontekstuelle miljøet og forsknings prosjektet via design-science aktiviteter (Hevner, 2007).

Rigor Cycle Rigorsyklusen relaterer design-science aktiviteter til tidligere kunnskap av vitenskapelige grunnlag, erfaring, og ekspertise. Rigorsyklusen sikrer innovasjon ved forsknings prosjektet, og grundig undersøkelse basert på kunnskap for å garantere at artefakten som produseres er forskningsbasert (Hevner, 2007).

Design Cycle Den sentrale designsyklusen looper mellom kjerneaktiviteter ved å bygge og evaluere artefakten og prosessen til forskningen (Hevner, 2007).

I dette prosjektet er artefakten og det endelige produktet et dokument som ble kalt «ProMat». Produktet er en samling av matematiske oppgaver og løsninger til disse oppgavene i form av algoritmer og programkoder. Innsatsen har vært å samle oppgaver som er til nytte for matematikklærere og hjelper dem å anvende programmering i matematikkfaget. Oppgavene måtte fylle kravene som var definert i starten av prosjektet i visjonsdokumentet som for eksempel at oppgavene blir valgt slik at de samsvarer kompetansemålene i de nye læreplaner. Det ble også tatt hensyn til kjerneelementene i matematikkfaget etter fagfornyelsen og hva som fører til dybdelæring hos elevene. Algoritmer er blitt til mot formålet å hjelpe lærerne til å kunne ta dem i bruk og skrive programkoder mens de følger disse algoritmer.

Prosjektet startet med rigorsyklusen, ved å finne eksisterende løsninger og relaterte litteraturer. Undersøkelse om tidligere arbeid, eksisterende løsninger og vitenskapelige forskninger viste at det er behov for en kategorisert løsning består av eksempel oppgaver fra matematikkfagene på VGS som viser fremgangsmåten fra problem til program.

Deretter kom prosjektet inn til designsyklusen, og arbeidet ble videre å velge de matematiske oppgaver som er engasjerende, interessante og som stimulerer kreativitet hos elevene samtidig som bidrar med dybdelæring. Oppgavene ble samlet blant de oppgavene som regnes også som viktige oppgaver og blant de temaene som er understreket i de nye læreplaner.

Oppgavene ble valgt på den måten å dekke de mest praktiske kompetansemålene som er relatert til de virkelige situasjoner i samfunnet. Det ble forsøkt å produsere et fullstendig produkt slik at lærerne kan hente relevante oppgaver i henhold til hvert matematikkfag og bruke dem direkte i klasserommet. Det vil si relevanssyklusen ble gjennomført parallelt med designsyklusen.

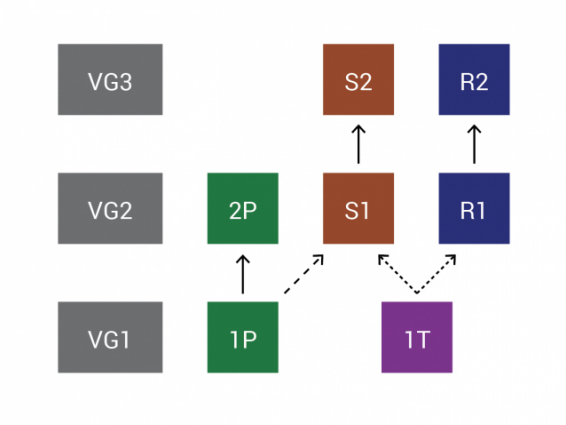
På grunn av begrenset tid og stor arbeidsmengde var det ikke mulig å teste artefakten med noen testbrukere for å sikre produktets kvalitet. Det var faktisk rimeligst å teste produktet, evaluere resultatet og hvis det var nødvendig, gå tilbake til designsyklusen, det vil si å loope mellom design- og relevanssyklusen. Dette er noe som nevnes som videre arbeid i siste kapittelet.

4 Resultater

På videregående skole kan elevene velge enten studieforberedende eller yrkesfaglige programmer. For studieforberedende finnes 5, og for yrkesfaglige 10 ulike utdannings programmer. Elevene kan fritt velge mellom disse utdannings programmene. Studieforberedende utdannings programmer varer tre år, og elevene etter fullføring av disse årene, får generell studiekompetanse som kvalifiserer dem til å fortsette ved universitet eller høyskole. Yrkesfaglige utdannings programmer varer som regel 4 år hvor elevene går to år på skole, og de to neste årene jobber de som lærling i en bedrift (Utdanning.no, udatert; Utdanningsdirektoratet, udatert).

Første året på VGS er matematikk obligatorisk, og elevene må velge mellom praktisk matematikk (1P) eller teoretisk matematikk (1T). Forskjellen mellom de to fagene er at P-matte dreier seg for det meste om praktiske situasjoner fra dagliglivet, samfunn og arbeidsliv, mens T-matte handler om teoretisk matematikk, og tema som algoritmisk tenkning og problemløsning er sentrale temaer i dette faget. Matematikk 1T passer for elever som vil ta realfag på Vg2 og Vg3, og planlegger å velge matematikk R eller matematikk S som programfag (vilbli.no, udatert); se figur 4.1.

I alle studieforberedende utdannings programmene er matematikk obligatorisk på Vg2 hvor elevene må velge å ta enten matematikk som fellesfag (matematikk 2P) eller som programfag (matematikk R1 eller matematikk S1). De som har hatt matematikk 1P på Vg1, velger matematikk 2P, og avslutter matematikk etter Vg2. Elevene som har hatt matematikk 1T, kan velge mellom R1 eller S1, og de har mulighet til å fortsette matematikk ved å ta R2 eller S2 på Vg3 (vilbli.no, udatert).



Figur 4.1: Matematikk på VGS

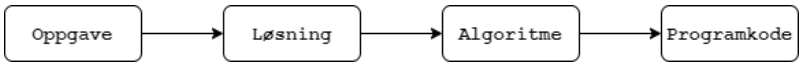
Source: (Utdanning.no, udatert)

Resultatet av dette prosjektet er et oppgavehefte består av oppgaver for matematikkfag på VGS[4]. Dokumentet begynnes med en kort introduksjon i Pythonprogrammering, og deles videre i 7 kapitler: 1P, 1T, 2P, R1, R2, S1, S2.

Hvert kapittel inneholder noen matematiske oppgaver som er blitt valgt blant temaer i kompetansemålene for tilhørende matematikkfaget. Samtidig ble det forsøkt å velge de oppgavene som regnes som interessante, engasjerende og viktige oppgaver.

Tanken bak oppgavene var å begynne med et problem og slutte med et program; se figur 4.2.

[4] Yrkesfaglige matematikkfag er ikke inkludert i dokumentet.



Figur 4.2: Fra problem til program

Hver oppgave begynnes med en beskrivelse av problemstillingen (oppgave), og deretter blir det forklart løsnings strategien til problemet (løsning); videre blir det vist fremgangsmåten for hvordan løsningen kan programmeres (algoritme), og til slutt legges til programkoden samt et eksempel output av programmet.

På grunn av stort antall oppgaver, blir noen av oppgavene nevnt som eksempler i dette kapittelet. For tilgang til alle oppgavene refereres leserne til Vedlegg A.

4.1 Vitenskapelige resultater

Basert på informasjon som ble forklart i kapittel 1 og kapittel 2, er det behov for utvikling og design av et dokument (produkt) som kan være til hjelp for matematikklærere på VGS. Til tross for flere tilbyr etterutdannings kurs for opplæring av programmering for matematikklærere, finnes det enda et behov for løsninger som lærerne kan bruke dem som en del av deres opplegg i klasserommet. Grunnen er at prosessen av å lære programmering (spesielt for dem som ikke har forkunnskap i programmering) vil ta tid, og dette betyr at lærerne vil ha utfordringer ved å anvende programmering i undervisning av deres fag.

Et kategorisert dokument med ferdige oppgaver som er også relevant til kompetansemålene i matematikkfagene vil hjelpe lærerne i begynnelsen av veien ved integrering av programmering i matematikk.

For å finne matematiske oppgaver, ble det først forsket om forskjellige kilder som finnes allerede som løsning av problemet. Oppgavene ble valgt både blant ressurser på Internett og fra trykte ressurser:

- Bøker
  - Programmering for matematikklærere av Henning Bueie
  - Programmering i skolen av Andreas Drolsum Haraldsrud, Henrik Andersen Sveinsson og Henrik Hillestad Løvold
  - Mønster: Matematikk 1T av Tove Kalvø mfl.
- Nettsider
  - NDLA
  - ProFag

Tabellen 4.1 viser en oversikt over det endelige resultatet.

Fag	Antall oppgaver	Tema
1P	5	tallregning, prosentregning, matematisk modellering og problemløsning, geometri
1T	8	programmering, algoritmisk tenkning og problemløsning, andregradslikninger, numeriske metoder, vekstfart og derivasjon
2P	5	likninger, prosentregning og vekstfaktor, økonomi
R1	4	numeriske metoder, derivasjon, vektorregning, regresjonsanalyse
R2	9	følger og rekker, integrasjon, differensiallikninger, modellering
S1	3	økonomi, sannsynlighet og kombinatorikk
S2	3	følger og rekker, eksponential vekst, statistikk

Tabell 4.1: Oversikt over resultater

Her kommer noen eksempler av disse oppgavene som resultat av prosjektet.

4.1.1 Euklidsk divisjon

Divisjons teorem (Euklidsk divisjon): Gitt to heltall a og b hvor $b \neq 0$, da finnes det to unike heltall q og r slik at:

$$a = bq + r \tag{4.1}$$

og

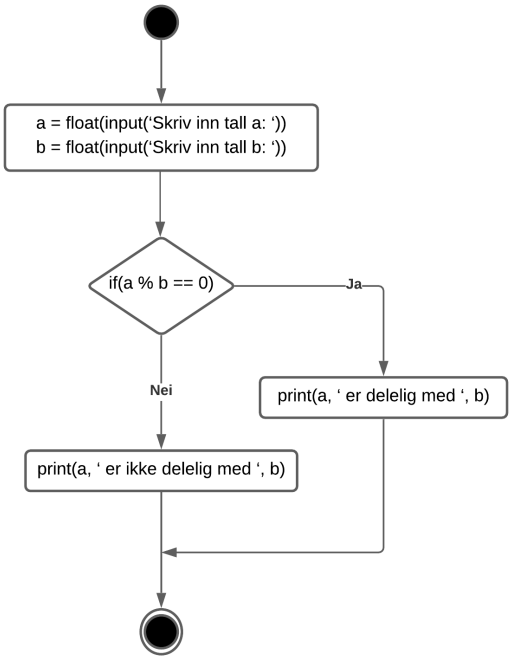
$$0 \leq r < |b|$$

Her kalles a *dividend*, b *divisor*, q *kvotient* og r *rest*.

Beregningen av kvotienten og resten fra a og b kalles divisjon eller euklidsk divisjon (I.T., udatert).

Oppgave Lag et program som tar to tall a og b, og sjekker om a er delelig med b.

Løsning Et tall er delelig med et annet tall dersom resten av divisjonen er lik null. I Python kan vi enkelt sjekke om delelighet med bruk av den aritmetiske operatoren modulus %. Et tall a er delelig med et tall b dersom $a\%b = 0$. Vi benytter dermed *if-else*-setningen for å programmere betingelsen for delelighet. Flytdiagrammet 4.3 viser algoritmen for å programmere oppgaven.



Figur 4.3: Flytdiagram for å sjekke om tall a er delelig med tall b.

Algoritme

1 Definere en funksjon *er_delelig()*.

1.1 Bruke *input()*-funksjonen for å få tallene a og b fra brukeren, samtidig som vi konverterer tallene til flyttall.

1.2 Hvis $a\%b$ er lik null

1.2.1 skrive ut at a er delelig med b.

1.3 Ellers

1.3.1 skrive ut at a ikke er delelig med b.

2 Kalle funksjonen *er_delelig()*.

```
'''
Programmet sjekker om a er delelig med b
'''

def er_delelig():

    # ber brukeren om å skrive inn tallene a og b, konverterer tallene til float
    a = float(input('Skriv inn tall a: '))
    b = float(input('Skriv inn tall b: '))

    # sjekker om a er delelig med b
    if a % b == 0:

        # skriver ut resultatet
        print(f'Tallet {a} er delelig med {b}')

    # ellers skriver ut det motsatte
    else:
        print(f'Tallet {a} er ikke delelig med {b}')

er_delelig()
```

Program 4.1: Programmet sjekker om et tall a er delelig med et annet tall b.

Eksempel output:

```
Skriv inn tall a: 4321
Skriv inn tall b: 1234
Tallet 4321.0 er ikke delelig med 1234.0
```

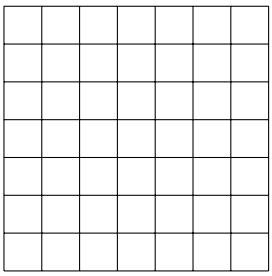
4.1.2 Kvadrattall

Kvadrattall er et figurtall på formen $S_n = n^2$, der n er et heltall (Weisstein, 2002c). De første kvadrattallene er:

$$\begin{aligned} 1^2 &= 1 \\ 2^2 &= 4 \\ 3^2 &= 9 \\ 4^2 &= 16 \\ 5^2 &= 25 \\ 6^2 &= 36 \\ &\vdots \end{aligned}$$

Vi kaller disse tallene *kvadrattall* i og med at slikt antall prikker eller kuler kan arrangeres som et geometrisk kvadrat, dvs. det n-te kvadrattallet har n prikker eller kuler i hver sin sidekant.

Oppgave Hvor mange kvadrater finnes det på figuren nedenfor?

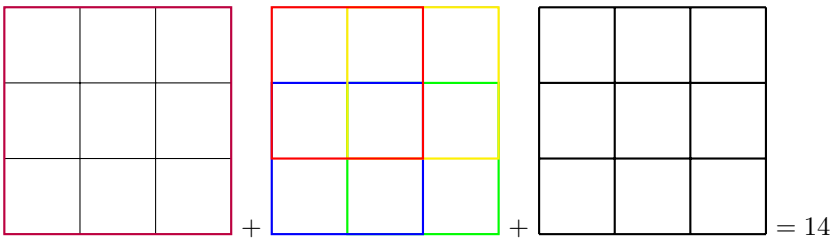


Oppgaven og problemløsnings strategien[5] er hentet fra (Kalvø mfl., 2020).

Problemløsnings strategi

- **Forstå problemet:** Denne figuren består av langt flere kvadrater enn de 49 små kvadratene. Vi kan tegne kvadrater som har 2 i høyde og bredde, vi kan tegne kvadrater som har 3 i høyde og bredde osv. Så vi må forstå at problemet er mer komplekst enn å bare telle de små kvadratene. Når vi har forstått problemet, går vi i gang med å lage en plan.
- **Lage plan:** Vi kan lage et kvadrat på 2×2, og flytte dette rundt for å telle hvor mange slike kvadrater man kan lage. Man kan også tenke at hun har 3×3, og flytte disse rundt, men dette vil kanskje ta veldig lang tid.

  Det lønner seg dermed å forenkle problemet, dvs. at vi lager et problem som er enklere å løse. Vi kan lage en figur som er delt inn i 2×2 kvadrater der har vi 4 små ruter og 1 stor rute, og til sammen $1 + 4 = 5$ kvadrater. Nå er vi på det steget som heter å gjennomføre planen.
- **Gjennomføre planen:** Vi deler inn kvadratet i 3×3 ruter, og teller hvor mange kvadrater vi kan lage.



  Vi har nå 1 stort kvadrat som er hele figuren, 4 kvadrater som er 2×2 kvadrater, og 9 små kvadrater. Vi får altså $1+4+9 = 14$ kvadrater. Det finnes en sammenheng i denne metoden; vi ser at 1, 4, og 9 er kvadrattallene. Når vi har 2×2 ruter får vi summen av de 2 første kvadrattallene, og når vi har 3×3 ruter får vi summen av de 3 første kvadrattallene. Det betyr at med 7×7 ruter, vil summen av kvadrater være lik summen av de 7 første kvadrattallene, dvs. $1 + 4 + 9 + 16 + 25 + 36 + 49 = 140$ kvadrater. Så kommer vi til det fjerde trinnet i problemløsnings strategien.
- **Se tilbake:** På dette trinnet skal vi bli sikre om at løsningen er riktig. Vi har vist at mønsteret vårt stemmer for figur med 1, 4 og 9 ruter. Vi kan se tilbake, og prøve med 4×4 ruter, og gjør vi det, ser vi at løsningen stemmer.

Vi bruker dette mønsteret for å programmere oppgaven.

Algoritme

1. Spørre brukeren om antall kvadrater for hver side.

[5] Problemløsnings strategien ble først definert av Pólya (1957).

2. Definere en variabel som kalles *resultat*, og sette den lik 0.
3. Vi kan videre bruke en *for*-løkke for å finne svaret. For hver rute $i = 1, 2, 3, ...$, så lange i ikke er lik antallet av kvadrater per side, er $resultat \mathrel{+}= i^2$.
4. Skrive ut resultatet.

Vi må være oppmerksomme på at *for*-løkken begynnes med 0, så vi må sette antall kvadrater lik $antall\_kvadrater + 1$. Alternativt kan vi skrive koden slik:

```
for i in (i+1 for i in range(antall_kvadrater)):
...
```

```
'''
Programmet finner antall kvadrater på et kvadratisk rutenett
'''

# spør brukeren om antall kvadrater per side
antall_kvadrater = int(input('Skriv inn antall kvadrat per side: '))

# definerer variabelen resultat
resultat = 0

# bruker for-løkke for å summere kvadrattallene
for i in range(antall_kvadrater + 1):
    resultat += i**2

# skriver ut resultatet
print('Antall kvadrater er ', resultat)
```

Program 4.2: Programmet finner antall kvadrater på et kvadratisk rutenett.

Output:

```
Skriv inn antall kvadrat per side: 7
Antall kvadrater er  140
```

4.1.3 Palindromtall

"Et palindromtall er et tall som forblir det samme når det skrives fremover eller bakover, dvs. er på formen $a_1a_2...a_2a_1$"(Weisstein, 2002b).

Oppgave Lag et program som finner hvor mange palindromtall som finnes i intervallet $[a, b]$ hvor a og b er input-verdiene gitt av brukeren.

Løsning For å løse oppgaven må vi lage en funksjon som sjekker om et tall er et palindromtall. For dette reverserer vi tallet og sjekker om tallet og det reverserte er like.

Problemløsnings strategien er hentet fra (ManBearPig, 2016).

Problemløsnings strategi Vi fjerner det siste sifferet fra tallet og legger det til revers-tallet, vi fortsetter til det opprinnelige tallet er borte, og det reverserte tallet er fullført. Metoden forutsetter at tallet er et heltall.

- **Steg 1: Isolere det siste sifferet**

  ```
  rest = tall % 10
  ```

  Restdivisjon-operatoren returnerer resten av en divisjon. I dette tilfellet deler vi tallet med 10 og returnerer resten, dvs. det siste sifferet. Vi lagrer dette tallet i et heltall-variabel kalt *rest*.

- **Steg 2: Legge rest til revers**

  ```
  revers = (revers * 10) + rest
  ```

  Vi begynner å bygge det reverserte tallet ved å legge *rest* i *revers*-variabelen. Vi multipliserer *revers* med 10, grunnen er at vi har delt tallet vårt på 10. Så for å få riktig tall, ganger vi resten med 10 hver gang vi henter den fra det opprinnelige tallet.

- **Steg 3: Fjerne det siste sifferet fra tallet**

  ```
  tall = tall // 10
  ```

  For å fjerne det siste sifferet fra tallet deler vi det med 10. Vi benytter *heltallsdivisjon*-operatoren som avrunder resultatet ned til nærmeste heltall f.eks. $244//10 = 24$

- **Gjenta steg 1-3**

  ```
  while (tall > 0)
  ```

  Gjenta denne prosessen til tallet er redusert til null og revers-tallet er fullført.

Algoritme

1 Definere en funksjon *er_palindromtall()*; funksjonen tar et heltall som parameter, og sjekker om tallet er et palindromtall eller ikke.

- 1.1 Definere en variable *revers*, og sette den lik 0.
- 1.2 Definere en variabel *temp*, og sette den lik tallet vi fikk som parameter. Denne variabelen bruker vi som en hjelpevariabel.
- 1.3 Benytte en *while*-løkke som looper og finner palindromtallene så lenge *temp* ikke er lik null.
- 1.4 Definere en variabel *rest*, og sette den lik *temp*%10.
- 1.5 Multiplisere *revers* med 10, og legge *rest* til den; vi setter *revers* lik dens nye verdi.
- 1.6 Bruke heltallsdivisjon, og dele *temp* på 10; vi setter *temp* lik dens nye verdi.
- 1.7 Sjekke om tallet og dets reverserte er like; funksjonen returnerer *True* dersom de er like, og *False* ellers.

2 Definere *main()*-funksjonen for å finne palindromtallene på intervallet $[a, b]$.

- 2.1 Få input-verdiene a og b fra brukeren; og konvertere dem til heltall.
- 2.2 Benytte en *for*-løkke som looper fra a til b.
  - 2.2.1 Sjekke om tallet på indeks i er et palindromtall ved å kalle *er_palindromtall()*-funksjonen.
    - 2.2.1.1 Skrive ut tallet hvis tallet er et palindromtall.

3 Kalle *main()*-funksjonen.

```
'''
Programmet finner palindromtall på et intervall [a, b]
'''
# definerer funksjon som sjekker om et tall er palindrom
def er_palindromtall(tall: int):

    # definerer en variabel for den reversen av tallet
    revers = 0

    # temp brukes som hjelpetall
    temp = tall

    # looper så lenge tallet er større enn 0
    while temp > 0:

        # deler tallet på 10, og finner resten
        rest = temp % 10

        # gang revers med 10 og legg rest til den
        revers = revers * 10 + rest

        # sett tallet lik tallet del på 10
        temp //= 10

    # hvis tallet er lik revers returnerer true, ellers false
    return (tall == revers)


# main-funksjonen skriver ut alle palindromtallene i intervallet [a, b]
def main():

    # får startverdi fra brukeren
    a = int(input('Oppgi startverdien: '))

    # får sluttverdi fra brukeren
    b = int(input('Oppgi sluttverdien: '))

    # looper fra a til b; legger 1 til b siden løkken begynnes fra indeks 0
    for i in range(a, b+1):

        # sjekker om tallet på indeks i er et palindromtall
        if er_palindromtall(i):

            # skriver ut tallet, end = ' ' setter et mellomrom etter tallet (neste tall kommer etter)
            print(i, end = ' ')

main()
```

Program 4.3: Programmet finner palindromtall på et intervall [a, b].

Eksempel output:

```
Oppgi startverdien: 100
Oppgi sluttverdien: 1000
101 111 121 131 141 151 161 171 181 191 202 212 222 232 242 252 262 272 282 292 303 313 323 333
↪  343 353 363 373 383 393 404 414 424 434 444 454 464 474 484 494 505 515 525 535 545 555 565
↪  575 585 595 606 616 626 636 646 656 666 676 686 696 707 717 727 737 747 757 767 777 787 797
↪  808 818 828 838 848 858 868 878 888 898 909 919 929 939 949 959 969 979 989 999
```

4.1.4 Derivasjon

Derivasjon er blant de matematiske begrepene som har flere regler og teknikker. Dette kan føre til at elever blir distrahert fra å forstå hva derivasjon faktisk er. Som løsning for dette kan numerisk derivasjon gi en bedre forståelse av hva vi gjør når vi deriverer en funksjon, i og med at numerisk derivasjon viser faktisk definisjonen for den deriverte (Haraldsrud mfl., 2020, s. 211).

Vi beregner den deriverte av en funksjon analytisk gitt ved formelen:

$$f'(x) = \lim_{\Delta x \to 0} = \frac{f(x + \Delta x) - f(x)}{\Delta x} \tag{4.2}$$

Vi kan beregne en tilnærming for denne grensen der Δx går mot 0 med en svært liten Δx. Denne metoden kalles *Newtons kvotient* (Haraldsrud mfl., 2020, s. 211):

$$f'(x) \approx \frac{f(x + \Delta x) - f(x)}{\Delta x} \tag{4.3}$$

Oppgave Finn den deriverte til funksjonen $f(x) = \sqrt{x}$ i punkt $x = 3$.

Løsning Bueie (2019, s. 100) bruker programkode 4.4 for å finne den deriverte til en gitt funksjon i et gitt punkt.

```
'''
Programmet finner den deriverte til f(x) numerisk i punkt x=3
'''

# definerer en funksjon f som returnerer vår funksjon
def f(x):
    return np.sqrt(x)

# definerer en funksjon som returnerer den deriverte av f
def derivert(x):

    # definerer steglengde
    delta_x = 0.001

    # finner endringer i y-verdier
    endring = f(x + delta_x) - f(x)

    # finner veksthastigheten ved å dele endringen i y på endringen i x
    vekst = endring/delta_x

    # returnerer veksthastigheten
    return vekst

# finner den deriverte av f in punkt x=3 med en steglengde=0.001
print(f'Den deriverte av funksjonen f i punkt x = 3 er lik {derivert(3):.3f}')
```

Program 4.4: Programmet finner den deriverte til $f(x)$ numerisk i punkt $x = 3$.

Output:

```
Den deriverte av funksjonen f i punkt x = 3 er lik 0.289
```

Vi kan utvide koden ved å legge til tegning av grafene $f(x)$ og $f'(x)$ samt å spørre brukeren om i hvilket punkt og med hvilken steglengde programmet skal finne den deriverte av f.

Algoritme

1 Importere *NumPy*-biblioteket, og *matplotlib.pyplot*-modulen.

2 Definere en funksjon som returnerer funksjonen $f(x) = \sqrt{x}$; funksjonen tar x som parameter.

3 Definere en funksjon kalt *derivert* som tar x og Δx som parametere.

3.1 Benytte formelen 4.3 for å finne den deriverte av f. Her kan vi enten definere en variabel *vekst* (Du kan kalle variabelen hva du ønsker) og finne den deriverte med en gang,

```
vekst = (f(x + delta_x) - f(x)) / delta_x
```

eller vi kan finne først telleren av formelen og så dele den på nevneren.

```
endring = f(x + delta_x) - f(x)
vekst = endring/delta_x
```

3.2 Returnere vekst.

4 Definere funksjonen *main()* som klientprogram.

4.1 Få x-verdien fra brukeren (Hvilket punkt programmet skal finne den deriverte for); konvertere input-verdien til heltall.

4.2 Spørre brukeren om steglengde. Denne størrelsen må defineres som *float* fordi som regel er det ønskelig å defineres små verdier for Δx.

4.3 Beregne den deriverte av $f(x)$ ved å kalle *derivert()*-funksjonen.

4.4 Skrive ut resultatet.

4.5 Spørre brukeren om start- og sluttpunkt på x-aksen samt antall punkter for tegning av grafer. Denne verdien er som standard lik 50. Konvertere input-verdiene til heltall.

4.6 Definere x-verdier ved å kalle *linspace()*-funksjonen fra *NumPy*-biblioteket. Sette de verdiene som er mottatt fra brukeren som parameter inn i funksjonen.

4.7 Definere y-verdier for $f()$-funksjonen. Funksjonen tar *x-verdier* som parameter.

4.8 Definere y-verdier for den deriverte av $f()$, $(f'(x))$ ved å kalle *derivert()*-funksjonen. Funksjonen tar *xverdier* og *delta_x* som parametere.

4.9 Sette størrelse for plottet ved å kalle *figure()*-funksjonen fra *pyplot*. Vi definerer figurstørrelse som parameter til funksjonen:

```
figure(figsize=(x,y))
```

4.10 Sette tittel for plottet ved å kalle *title()*-funksjonen fra *pyplot*.

4.11 Sette label for x- og y-aksen ved å kalle henholdsvis *xlabel()*- og *ylabel()*-funksjonen.

4.12 Kalle *plot()*-funksjonen for å tegne grafer for $f(x)$ og $f'(x)$. Vi kan kalle funksjonen to ganger og hver gang sette inn x- og y-verdier for funksjonen, eller vi kan kalle *plot()*-funksjonen én gang og sette x- og y-verdier for begge funksjoner etter hverandre. Det er også mulig å definere forskjellige farger for grafer ved å skrive fargekode som streng etter x og y:

```
plot(x_verdier, y_verdier,'r')
```

her står 'r' for rødfarge.

For å vise punktet som brukeren ønsker å finne den derivert i, setter vi x og y-derivert inn i *plot()*-funksjonen. For punktet bruker vi 'o' eller '*farge* o' som parameter etter x- og y-verdi.

4.13 Vise koordinater for det deriverte punktet på plottet ved hjelp av *text()*-funksjonen fra *pyplot*-modulen. Funksjonen tar x- og y-verdi og teksten som parametere. Syntaksen for å skrive koordinater som tekst er som følger:

```
text(x, y,'({},␣{})'.format(x, y))
```

4.14 Kalle *legend()*-funksjonen og skrive navnene til funksjoner (som en liste av strenger) som parameter.

4.15 Sette rutenett på plottet ved å kalle *grid()*-funksjonen.

4.16 Vise plottet ved å kalle *show()*-funksjonen.

5 Kalle *main()*-funksjonen.

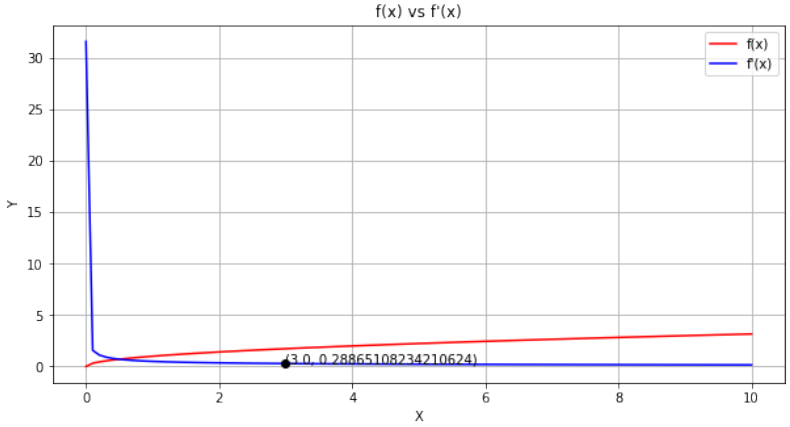
```
'''
utvidet kode for numerisk derivasjon, programmet tegner grafen til f(x) og f'(x),
og finner den derivert til f(x) i x=x_i grafisk
'''

# importerer biblioteker
import matplotlib.pyplot as plt
import numpy as np

# definerer en funksjon f som returnerer vår funksjon
def f(x):
    return np.sqrt(x)

# definerer en funksjon som returnerer den deriverte av f
def derivert(x, delta_x):

    endring = f(x + delta_x) - f(x) # finner endringer i y-verdier
    vekst = endring/delta_x   # finner veksthastigheten ved å dele endringen i y på endringen i x

    return vekst  # returnerer veksthastigheten

# funksjonen for å teste metoden samt tegning av grafer
def main():
    '''
    data for beregning av f'(x)
    '''
    # får x-verdien og steglengde fra brukeren
    x = float(input('Skriv inn x-verdien: '))
    delta_x = float(input('Skriv inn steglengde: '))

    # kaller derivert()-metoden for å finne den deriverte av f
    f_derivert = derivert(x, delta_x)

    # skrive ut resultatet
    print(f'\nDen deriverte av funksjonen f i punkt x = {x} er lik {f_derivert:.3f}\n')

    '''
    data for tegning av grafer
    '''
    # får startpunkt, sluttpunkt og antall punkter for x-aksen fra brukeren
    x_start = int(input('Skriv inn startverdi: '))
    x_slutt = int(input('Skriv inn sluttverdi: '))
    antall_punkter = int(input('Skriv inn antall sampler: '))

    # definerer x-verdier for grafen
    x_verdier = np.linspace(x_start, x_slutt, antall_punkter)

    # definerer y-verdier for f(x) og f'(x)
    y_verdier = f(x_verdier)
    y_derivert = derivert(x_verdier, delta_x)

    plt.figure(figsize=(10,5))   # setter størrelse for plottet

    # setter tittel for plottet, og label for aksene
    plt.title('f(x) vs f\'(x)')
    plt.xlabel('X')
    plt.ylabel('Y')

    # tegner grafen for f(x) og f'(x)
    plt.plot(x_verdier, y_verdier, 'r', x_verdier, y_derivert, 'b', x, f_derivert, 'ko')
    # viser koordinat for derivert punkt på plottet
    plt.text(x, f_derivert, '({}, {})'.format(x, f_derivert))

    plt.legend(['f(x)', 'f\'(x)'])  # legger til label for funksjoner
    plt.grid()  # setter rutenett på plottet
    plt.show()  # viser plottet

main()
```

Program 4.5: Programmet tegner grafen til $f(x)$ og $f'(x)$, og finner den derivert til $f(x)$ i $x = x_i$ grafisk.

Output:

```
Programmet finner den deriverte av f(x)

Skriv inn x-verdien: 3
Skriv inn steglengde: .001

Den deriverte av funksjonen f i punkt x = 3.0 er lik 0.289

Nå tegner grafen til f(x) og f'(x)

Skriv inn startverdi: 0
Skriv inn sluttverdi: 10
Skriv inn antall sampler: 100
```

Tilnærmet verdi for den deriverte av $f(x) = \sqrt{x}$ i punkt $x = 3$, med tre desimaler nøyaktighet er 0.289. Figur 4.4 viser også den samme løsningen grafisk.



Figur 4.4: Grafisk løsning av $f(x) = \sqrt{x}$ i punkt $x = 3$.

4.1.5 Integrasjon

Begrepet integral refereres som regel til summering av uendelige stykker for å finne innholdet i en kontinuerlig region. Prosessen med å beregne en integral kalles integrasjon, og den tilnærmede beregningen av en integral kalles numerisk integrasjon (Weisstein, 2002a).

Oppgave Finn en numerisk tilnærming for det bestemte integralet ved hjelp av Rektangelmetoden.

$$\int_2^5 \sqrt{x}\, dx \tag{4.4}$$

Løsning Når vi skal løse bestemte integraler numerisk, kan vi regne ut arealet under en funksjonsgraf med utgangspunkt i en funksjon.

Fremgangsmåten for rektangelmetoden er å dele arealet under grafen i flere rektangler, og finne summen av disse rektanglene; desto flere rektangler setter vi inn, desto mer nøyaktig svar får vi. Vi benytter formelen for rektangelmetoden A.47 for å løse oppgaven.

Vi har allerede verdien til a og b, det som ikke er oppgitt, er antall rektangler n. Denne verdien skal vi få fra brukeren, deretter kan vi regne ut bredden h til rektangler. Lengden for hvert rektangel

er lik $f(x_i)$ der $i = 1, 2, 3, \quad ..., n$. Summen av arealet til alle de n rektanglene er det bestemte integralet av $f(x) = \sqrt{x}$ fra $a = 2$ til $b = 5$. Se figur 4.5 som illustrerer bruk av rektangelmetoden med 10 rektangler for å løse integralet.

Alternativt kan vi løse oppgaven ved å bruke *sum()*-funksjonen fra *NnmPy*-biblioteket. Vi løser oppgaven med begge metodene, også legger vi til et plott som viser rektangelmetoden grafisk.[6]

Algoritme

1 Importere biblioteket *Numpy*, og *matplotlib.pyplot*-modulen.

2 Definere en funksjon $f()$ som returnerer $\sqrt{x}$.

3 Definere en funksjon *rektangelmetoden()*.

- 3.1 Få antall rektangler n fra brukeren; konvertere input-verdien til heltall.
- 3.2 Definere variablene a og b med de gitte verdiene ($a = 2$, $b = 5$).
- 3.3 Definere en variabel h som bredde for hvert rektangel. For dette bruker vi formelen A.46.
- 3.4 Definere en variabel *total_sum* for å lagre resultatet i.
- 3.5 Bruke en *for*-løkke som tar *range()*-funksjonen med n som parameter.
  - 3.5.1 Definere x lik $a + i * h$ der a er startpunktet av intervallet, og i indeksen for x.
  - 3.5.2 Beregne areal for hvert rektangel gitt ved lengden $f(x)$ $\times$ bredden h. Vi summerer arealene fortløpende mens vi looper gjennom løkken.
- 3.6 Skrive ut resultatet.
- 3.7 Definere og lage en liste for x-verdier ved å bruke *linspace()*-funksjonen. Funksjonen tar a, $b - h$ og n som parametere. (b er ikke inkludert i listen siden vi beregner venstre Riemann-sum).
- 3.8 Kalle funksjonen *sum()* fra *NumPy*; funksjonen returnerer summen av arealene til alle rektanglene. I vårt tilfelle er arealet lik $f(x\_verdier) * h$.
- 3.9 Skrive ut resultatet.
- 3.10 Definere og lage en liste for y-verdier; vi kaller *f()*-funksjonen og setter x-verdier som parameter for funksjonen.
- 3.11 Plotte grafen til $f(x)$ ved hjelp av *plot()*-funksjonen. Funksjonen tar x- og y-verdier som parametere. Det er også mulig å sette farge til grafen ved å skrive farge som streng.
- 3.12 Lage liste over venstre x- og y-verdier; for eksempel lager vi listen for venstre x-verdier med følgende syntaks:

```
x_venstre = x_verdier[:-1]
```

- 3.13 Sette tittel for plottet ved å kalle *title()*-funksjonen fra *pyplot*.
- 3.14 Sette label for x- og y-aksen ved å kalle henholdsvis *xlabel()*- og *ylabel()*-funksjonen.
- 3.15 Kalle *bar()*-funksjonen fra *Pyplot* for å plotte rektangler; funksjonen tar følgende parametere (Hunter mfl., 2021e):
  - **x:** x-verdier
  - **height:** y-verdier
  - **width:** h, bredde for rektangler
  - **alpha:** bestemmer gjennomsiktighets graden
  - **align:** justerer stolpene, standard er 'center', med 'edge' tar vi venstre kant på stolpene
  - **edgecolor:** farge til kantene

[6] Løsningsforslaget for denne delen er hentet fra (Walls, 2019b).

3.16 Vise plottet ved å kalle *show()*-funksjonen.

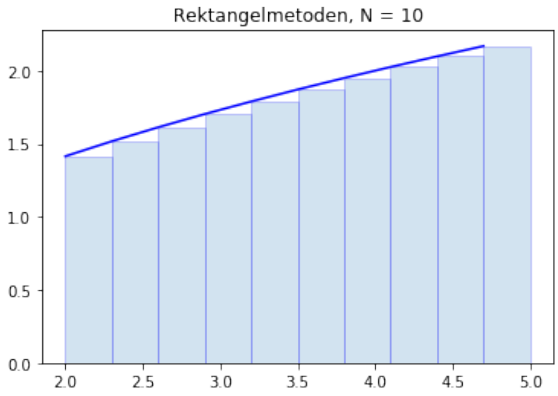
4 Kalle *rektangelmetoden()*.

```
'''
Programmet implementerer Rektangelmetoden (venstre Riemann-sum)
og løser funksjonen f(x) vha. metoden
'''

# importerer biblioteker
import matplotlib.pyplot as plt
import numpy as np

# funksjon som skal integreres
def f(x):
    return np.sqrt(x)

# definerer funksjonen for rektangelmetoden
def rektangelmetoden():
    n = int(input('Skriv inn antall rektangler: ')) # får verdien for antall rektangler

    # setter opp innstillinger
    a = 2                       # startverdi
    b = 5                       # sluttverdi
    h = (b - a)/n               # bredde
    total_sum = 0               # variabel for å lagre total summen

    for i in range(n): # vi looper gjennom løkken fra 0 til n
        x = a + i * h  # verdien av x_i er lik startverdi + i * bredde

        # areal av en rektangel er lik lengde * bredde her f(x_i) = lengde for i-te rektangelet
        areal = f(x) * h

        # summerer areal for hver rektangel inn i variabelen total_sum
        total_sum += areal

    # skriver ut resultatet
    print(f'Numerisk verdi av integralet (beregnet av egendefinert funksjon) er lik {total_sum:.3f}')

    '''
    Alternativ metode samt tegning av grafen:

    Vi lager et sett av x- og y-verdier, disse verdiene brukes for både tegning av grafen og beregning av summen
    x_verdier lager vi ved å kalle linspace()-funksjonen;
    b er ikke inkludert i x-verdier siden vi beregner venstre Riemann-sum
    '''
    x_verdier = np.linspace(a,b-h,n)
    '''
    kaller funksjonen sum() fra NumPy, funksjonen returnerer summen av arealene til alle rektangler
    her  f(x_verdier) * h = arealet til rektangler
    '''
    venstre_riemann_sum = np.sum(f(x_verdier) * h)

    # skriver ut summen
    print(f'Numerisk verdi av integralet (beregnet av sum()-funksjonen fra NumPy) er lik {total_sum:.3f}')

    y_verdier = f(x_verdier)  # lager en liste av y-verdier
    plt.plot(x_verdier,y_verdier, 'blue')  # tegner grafen til f

    # lager liste over venstre x- og y-verdier
    x_venstre = x_verdier[:-1]
    y_venstre = y_verdier[:-1]

    # setter tittel for grafen, format()-funksjonen benyttes for å skrive ut n-verdien på grafen
    plt.title('Rektangelmetoden, N = {}'.format(n))

    # setter label for x- og y-aksen
    plt.xlabel('x')
    plt.ylabel('y')
    '''
    bar-funksjonen tar følgende parametere:
    matplotlib.pyplot.bar(x, height, width=0.8, bottom=None, *, align='center', data=None, **kwargs)[source]
    x = x-verdier
    height = y_verdier
    width = h
    alpha = bestemmer gjennomsiktighetsgraden
    align = justerer stolpene, standard er 'center'. med 'edge' tar vi venstre kant på stolpene etter x-verdier
    edgecolor = farge til kantene
    '''
    plt.bar(x_verdier,y_verdier, h, alpha=0.2, align='edge',edgecolor='blue')
    plt.show()  # viser plottet

rektangelmetoden()
```

Program 4.6: Programmet implementerer Rektangelmetoden (venstre Riemann-sum).

Output:

```
Skriv inn antall rektangler: 10
Numerisk verdi av integralet (beregnet av egendefinert funksjon) er lik 5.444
Numerisk verdi av integralet (beregnet av sum()-funksjonen fra NumPy) er lik 5.444
```



Figur 4.5: Illustrasjon for bruk av rektangelmetoden for å løse integralet 4.4.

4.1.6 Eulers metode

Vi bruker Eulers metode for å finne en tilnærmet løsning for en første ordens differensiallikning. Metoden kan brukes for differensiallikninger av høyere orden også, men vi må først omforme likningen til et system består av første ordens likninger. Generelt kan en differensiallikning av orden n gjøres om til n likninger av orden 1.

Vurder en første ordens differensialligning med en initialtilstand:

$$y' = f(y,t) \ , \quad y(t_0) = y_0 \tag{4.5}$$

Eulers metode tar utgangspunkt i en startverdi y_n, og finner en tilnærmet verdi for y_{n+1}:

$$y_{n+1} = y_n + f(y_n, t_n)(t_{n+1} - t_n) \tag{4.6}$$

Vi bruker denne formelen for å implementere en funksjon *Eulers_metode()* i Python. Funksjonen tar en differensiallikning $f(y,t)$, y_0, og t-verdier som parametere, og returnerer en liste av y-verdier. De verdiene er som sagt tilnærminger til den eksakte løsningen. Programkoden for implementasjon av Eulers metode er lånet fra Walls (2019a).

I delkapittelet om første ordens differensiallikninger (Se 7.3.1 i oppgaveheftet.) så vi på likningen

$$\frac{dy}{dt} = -y$$

Likningen hadde en generell løsning lik

$$y = Ce^{-t}$$

Hvis vi setter en initialverdi $y(0) = 1$, får vi:

$$f(y,t) = e^{-t}$$

La løse differensiallikningen numerisk ved å bruke Eulers metode.

Algoritme for Eulers metode

1 Definere en funksjon *Eulers_metode()*; metoden tar en funksjon $f()$, initialverdi y_0 og t-verdier som parametere.

1.1 Definere en liste (matrise) for å lagre y-verdier, ved å kalle *zeros()*-funksjonen fra *NumPy*-biblioteket. Funksjonen tar et heltall n (eller en *tuple* $[n, n]$) som parameter og returnerer en $n \times n$ matrise. Her setter vi n lik lengden for t-verdier.

1.2 Sette initialverdien y_0 lik startverdi i listen for y-verdier (på indeks 0).

1.3 Bruke en *for*-løkke som looper fra 0 til (lengde til t-verdier)-1; siden vi starter fra 0, trekker vi 1 fra lengden til t-verdi-listen.

1.3.1 Bruke formelen 4.6, og finne y_{n+1}.

1.4 Returnere y.

```
'''
Programmet implementerer Eulers metoden (lånet fra github-siden til Patrick Walls)
parametere: funksjon f(y, t), initialverdi y0, tidsverdier
returnerer en liste av y-verdier
'''
def Eulers_metode(f, y0, t):

    '''
    lager en matrise (liste) for y-verdier;
    alle verdiene er lik 0;
    lengden på listen er lik lengden for t-verdier
    '''
    y = np.zeros(len(t))

    # setter initialverdien y0 lik startverdi i listen (på indeks 0)
    y[0] = y0

    # looper fra 0 til (lengde til t-verdier)-1 (siden vi starter fra 0)
    for n in range(0, len(t)-1):

        # finner y_n+1 i hver loop med hjelp av formelen for Eulers metode
        y[n+1] = y[n] + f(y[n], t[n]) * (t[n+1] - t[n])
    return y
```

Program 4.7: Programmet implementerer Eulers metode.

Algoritme for eksempel bruk av Eulers metode

1 Importere biblioteket *Numpy*, og modulen *pyplot*.

2 Definere en funksjon $f(y,t)$ som returnerer funksjonen $\frac{dy}{dt} = -y$.

3 Definere en liste for t-verdier ved å kalle *linspace()*-funksjonen fra *Numpy*-biblioteket. Vi velger et tidsintervall fra 0 til 10, med 20 punkter.

4 Definere initialverdi y_0, og sette den lik 1.

5 Beregne tilnærmede y-verdier ved å kalle *Eulers_metode()*-funksjonen. Vi lagrer disse i en variabel kalt *y_approx*.

6 Beregne eksakte y-verdier ved å mate den eksakte løsningen $f(y,t) = e^{-t}$ med t-verdier. Vi lagrer disse i en variabel kalt *y_eksakt*.

7 Sette tittel for plottet ved å kalle *title()*-funksjonen fra *pyplot*.

8 Sette label for x- og y-aksen ved å kalle henholdsvis *xlabel()*- og *ylabel()*-funksjonen.

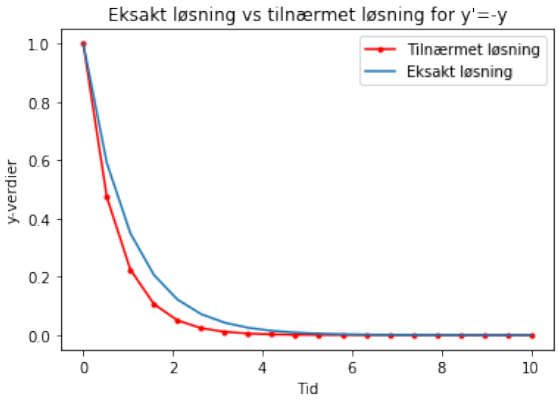
9 Plotte grafen for den eksakte løsningen mot den tilnærmede løsningen, for dette bruker vi *plot()*-funksjonen fra *matplotlib.pyplot*-modulen.

10 Sette label for grafene ved å kalle *legend()*-funksjonen; funksjonen tar en liste av strenger som parameter.

11 Vise plottet ved hjelp av *show()*-funksjonen.

```
'''
Programmet finner tilnærmet løsning for y'=-y vha. Eulers metode
'''

# importerer biblioteker
import numpy as np
import matplotlib.pyplot as plt

# definerer en funksjon f(y, t) som returnerer vår funksjon
def f(y, t):
    dydt = -y
    return dydt

# lager en liste for t-verdier
t_verdier = np.linspace(0, 10, 20)

# initialverdi for y
y0 = 1

# kaller Eulers_metode() for å finne tilnærmede y-verdier
y_approx = Eulers_metode(f, y0, t_verdier)

# lager en liste av eksakte y-verdier
y_eksakt = np.exp(-t_verdier)

# setter tittel, og label for x- og y-aksen
plt.title("Eksakt løsning vs tilnærmet løsning for y'=-y")
plt.xlabel('Tid')
plt.ylabel('y-verdier')

# plotter grafen for den eksakte løsningen mot den tilnærmede løsningen
plt.plot(t_verdier, y_approx, 'r.-', t_verdier, y_eksakt)

# setter label for grafer
plt.legend(['Tilnærmet løsning','Eksakt løsning'])

# viser plottet
plt.show()
```

Program 4.8: Programmet finner tilnærmet løsning for $y' = -y$ vha. Eulers metode.



Figur 4.6: Grafen illustrerer eksakt løsning mot tilnærmet løsning for likningen $y' = -y$.

Grafen 4.6 viser at Eulers metode har avvik fra den eksakte løsningen på tidspunkter fra 0 til 5, men etter på blir tilnærmingene veldig nære til de eksakte verdier.

Hvis vi velger steglengde for t-verdier så liten som mulig, vil feilen bli mindre.

Oppgave I denne oppgaven skal vi modellere populasjonen til rein. Vi vet at:

- Vi har en reinstamme med 300 dyr.
- Bæreevnen er 200 dyr.
- Naturlig vekst er 10% per år.
- Vekstfarten er gitt ved:

$$y' = 0.10y * (1 - \frac{y}{200}) \tag{4.7}$$

Lag en graf som viser utviklingen til reinpopulasjonen de første 20 årene. Finn ut når tallet på rein er ca. 250, kommenter hva du ser.

Oppgaven er hentet fra (ProFag, 2021).

Løsning Ved tidspunkt $t = 0$ har vi 300 dyr, altså $y_0 = 300$. Oppgaven er å vise utviklingen for de 20 første årene, dvs. vi skal vise utviklingen på tidsintervallet $[0, 20]$. Vi bruker *linspace()*-funksjonen for å lage en liste for tidsverdier, vi setter startpunkt lik 0 og sluttpunkt lik 20 med 20 punkter.

Vi løser oppgaven både ved å bruke *odeint()*-funksjonen og ved å bruke *Eulers_metode()*-funksjonen. Så sammenligner vi svarene vi får fra de to metodene. Vi plotter også grafer til begge metodene.

Algoritme

1 Importere biblioteket *Numpy*, modulen *matplotlib.pyplot*, og funksjoner *solve()*, *odeint()*, og *Table()* fra henholdsvis *sympy*-biblioteket, og *scipy.integrate*- og *astropy.table*-modulen.

2 Definere en funksjon $f(y, t)$ som returnerer funksjonen 4.7.

3 Lage en liste for tidsverdier ved å kalle *linspace()*-funksjonen; funksjonen tar startverdi, sluttverdi og antall punkter som parametere.

4 Definere en variabel y_0, og sette den lik 300.

5 kalle *Eulers_metode()* for å finne tilnærmede y-verdier, funksjonen tar f, y_0 og t-verdier som parametere.

6 Kalle *odeint()*-funksjonen for å finne eksakte y-verdier; funksjonen tar f, y_0 og t-verdier som parametere.

7 Finne koeffisienter til funksjonen som passer y-verdier ved å kalle *polyfit()*-funksjonen fra *NumPy*-biblioteket. Funksjonen tar x- og y-verdier, og polynomgraden som parametere. Vi velger polynomgraden lik 2.

8 Finne funksjonen $y(t)$ ved å kalle *poly1d()*-funksjonen; funksjonen tar koeffisientene og variabelen t som parametere, og returnerer funksjonen $y(t)$. Syntaksen er som følger:

```
poly1d(koeff[:,0], variable='t')
```

9 Skrive ut funksjonen $y(t)$.

10 Sette størrelse for plottet ved å kalle *figure()*-funksjonen fra *pyplot*. Vi definerer figurstørrelse som parameter til funksjonen:

```
figure(figsize=(x,y))
```

11 Sette tittel for plottet ved å kalle *title()*-funksjonen fra *pyplot*.

12 Sette label for x- og y-aksen ved å kalle henholdsvis *xlabel()*- og *ylabel()*-funksjonen.

13 Plotte grafen for den eksakte løsningen mot den tilnærmede løsningen ved å kalle *plot()*-funksjonen.

Vi kan sette label for grafen også. Syntaksen kan se ut som følger:

```
plot(x, y,'r.-', label='label')
```

14 Plotte linjen for $y = 250$, denne linjen vil hjelpe oss for å finne når reinpopulasjonen er lik 250. For å plotte horisontal linje benytter vi funksjonen *xhline()* fra *pyplot*-modulen. Vi setter y-verdi, farge og label til grafen som parametere. Det er også mulig å definere intervall for grafen (Hunter mfl., 2021d):

```
axhline(y=0, xmin=0, xmax=1, **kwargs)
```

15 Sette label for grafer ved å kalle *legend()*-funksjonen.

16 Sette rutenett for plottet ved å kalle *grid()*-funksjonen.

17 Vise plottet ved å kalle *show()*-funksjonen.

18 Sette verdier i en tabell for å finne ut når reinpopulasjonen er 250. For dette kaller vi funksjonen *Tabell()*; funksjonen tar flere parametere (Astropy-Developers, 2021):

```
Table(data=None, masked=False, names=None, dtype=None, meta=
    None, copy=True, rows=None, copy_indices=True, units=None,
    descriptions=None, **kwargs)
```

Som data setter vi inn tidsverdier, og eksakte og tilnærmede verdier. På grunn av verdier har flere desimaler, avrunder vi verdiene ved å kalle *round()*-funksjonen fra *Numpy* siden vi har Numpy-arrayer. Vi setter også navn til hver kolonne som parameter. Syntaksen kan se slik ut:

```
Table([np.round(t_verdier, decimals=0), np.round(y_eksakt,
    decimals=0), np.round(y_approx, decimals=0)],names=(' Aar',
    'Eksakt verdi','Tilnaermet verdi'))
```

19 Skrive ut tabellen.

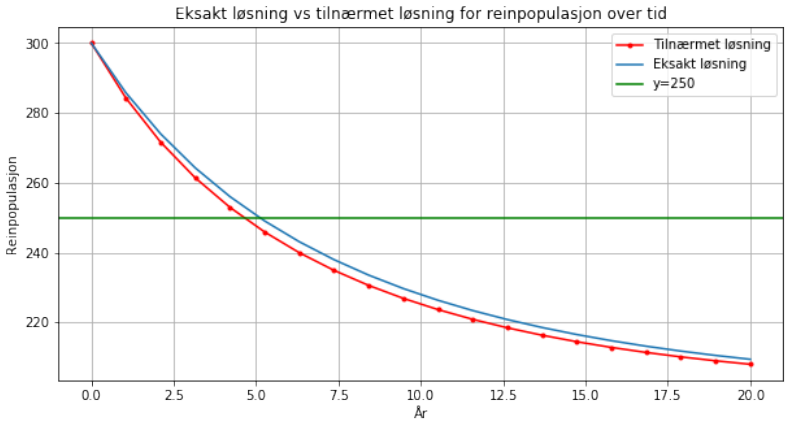
```
'''
programmet finner utviklingen av reinpopulasjon over tid
'''

# importerer biblioteker
import numpy as np
from scipy.integrate import odeint
from sympy import solve
import matplotlib.pyplot as plt
from astropy.table import Table

# definerer en funksjon f(y, t) som returnerer vår funksjon
def f(y, t):
    dydt = 0.10*y * (1-y/200)
    return dydt

# lager en liste for t-verdier
t_verdier = np.linspace(0, 20, 20)

# initialverdi for y
y0 = 300

# kaller Eulers_metode() for å finne tilnærmede y-verdier
y_approx = Eulers_metode(f, y0, t_verdier)

'''
løser differensiallikningen ved å kalle på odeint()-funksjonen
parametere: funksjonen f(y, t), initialverdi y0, t_veridier
returnerer en liste av y-verdier (eksakt løsning)
'''
y_eksakt = odeint(f, y0, t_verdier)

'''
finner koeffisienter til en funksjon som passer y-verdier (velger polynom av grad 2)
koeffisientene er en nx1 matrise avhengig av polynomgraden
'''
koeff = np.polyfit(t_verdier, y_eksakt, 2)

'''
finner funksjonen y(t) ved å kalle på poly1d()-funksjonen
parametere: koeffisienter, variabel
returnerer funksjonen y(t)
'''
y = np.poly1d(koeff[:,0], variable='t')

# skriver ut funksjonen
print('y(t)=\n', y)

# setter størrelse for plottet
plt.figure(figsize=(10,5))

# setter tittel, og label for x- og y-aksen
plt.title("Eksakt løsning vs tilnærmet løsning for reinpopulasjon over tid")
plt.xlabel('År')
plt.ylabel('Reinpopulasjon')

# plotter grafen for den eksakte løsningen mot den tilnærmede løsningen
plt.plot(t_verdier, y_approx, 'r.-', label='Tilnærmet løsning')
plt.plot(t_verdier, y_eksakt, '-', label='Eksakt løsning')
plt.axhline(250, color='green', label='y=250')

# setter label for grafer
plt.legend()

# setter rutenett
plt.grid()

# viser plottet
plt.show()

# setter verdier i en tabell for å finne ut når reinpopulasjonen er 250
tabell = Table([np.round(t_verdier, decimals=0), np.round(y_eksakt, decimals=0), np.round(y_approx, decimals=0)],
               names=('År', 'Eksakt verdi', 'Tilnærmet verdi'))

# skriver ut tabellen
print(tabell)
```

Program 4.9: Programmet finner utviklingen av reinpopulasjon over tid både analytisk og numerisk, og plotter de to løsningene mot hverandre.

Output:

```
y(t)=
          2
0.2624 t - 9.271 t + 293.7
```

År	Eksakt verdi	Tilnærmet verdi
0.0	300.0	300.0
1.0	286.0	284.0
2.0	274.0	272.0
3.0	264.0	261.0
4.0	256.0	253.0
5.0	249.0	246.0
6.0	243.0	240.0
7.0	238.0	235.0
8.0	234.0	231.0
9.0	230.0	227.0
11.0	226.0	224.0
12.0	223.0	221.0
13.0	221.0	218.0
14.0	219.0	216.0
15.0	217.0	214.0
16.0	215.0	213.0
17.0	213.0	211.0
18.0	212.0	210.0
19.0	211.0	209.0
20.0	209.0	208.0



Figur 4.7: Plottet viser utviklingen av reinpopulasjon over 20 år.

Vi ser både fra tabellen og fra grafen 4.7 at tilnærmede verdier er litt forskjellige fra de eksakte verdier. Tabellen viser at reinpopulasjonen i år 5 er lik 249 blant de eksakte verdier, som er den nærmeste verdien til 250, mens blant de tilnærmede verdier er den nærmeste verdien til 250, i år 4 lik 253. Dersom vi øker antall punkter for t-verdier dvs. hvis vi velger Δt mindre, vil den nærmeste verdien for begge deler (tilnærmede og eksakte verdier) være i år 5.

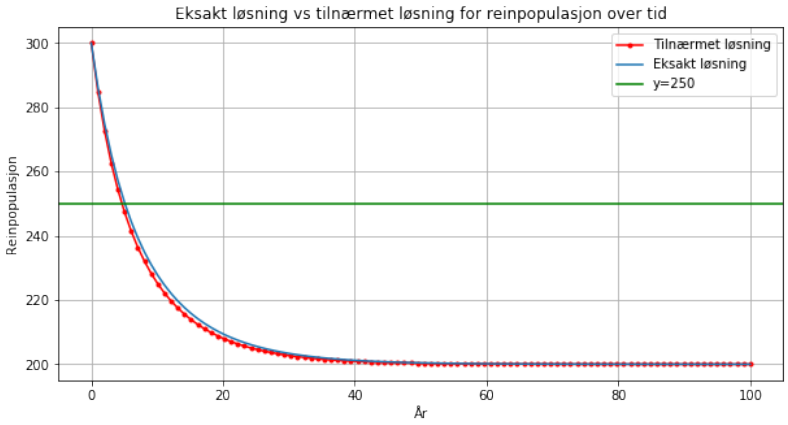
"Reinpopulasjonen avtar raskt de første årene. Etter ca. 15 år, når populasjonen kommer under 215 rein, avtar den mye saktere.

Hvis vi plotter utviklingen av populasjonen over et bredere intervall, for eksempel de første 100 årene, ser vi at etter $40 - 50$ år begynner reinpopulasjonen å stabilisere seg på 200 rein, som er bæreevnen til stammen"(ProFag, 2021).

Vi endrer tidsintervallet, og løser likningen på nytt og ser at konklusjonen stemmer.

Output:

```
y(t)=
          2
0.01424 t - 1.841 t + 252.9

År  Eksakt verdi        Tilnærmet verdi
----- ---------------- ---------------
  0.0            300.0           300.0
  1.0            286.0           285.0
  2.0            275.0           273.0
  3.0            265.0           263.0
  4.0            257.0           254.0
  5.0            250.0           247.0
  6.0            244.0           241.0
  7.0            239.0           236.0
  8.0            235.0           232.0
  9.0            231.0           228.0
  ...              ...             ...
 90.0            200.0           200.0
 91.0            200.0           200.0
 92.0            200.0           200.0
 93.0            200.0           200.0
 94.0            200.0           200.0
 95.0            200.0           200.0
 96.0            200.0           200.0
 97.0            200.0           200.0
 98.0            200.0           200.0
 99.0            200.0           200.0
100.0            200.0           200.0
Length = 100 rows
```



Figur 4.8: Plottet viser utviklingen av reinpopulasjon over 100 år.

Grafen 4.8 viser at utviklingen av reinpopulasjon stabiliserer seg på $y = 200$ som er bæreevnen til reinpopulasjonen.

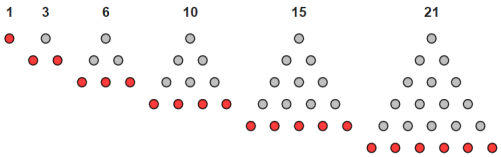
4.1.7 Kombinatorikk

"Trekanttallet T_n er et figurantall som kan representeres i form av et trekantet rutenett av noder der den første raden inneholder et enkelt element og hver påfølgende rad inneholder ett mer element enn det forrige"(Weisstein, 2002d). De trekanttallene er derfor:

$\Delta_1 \to 1$
$\Delta_2 \to 1 + 2 = 3$

$\Delta_3 \rightarrow 1+2+3=6$
$\Delta_4 \rightarrow 1+2+3+4=10$
.
.
.
$\Delta_n \rightarrow 1+2+3+4+(n-1)+n=\frac{1}{2}n(n+1)$

La T_n være trekanttall nummer n. Vi ser at hvert trekanttall T_n er summen av det forrige trekanttallet T_{n-1} pluss n prikker.



Figur 4.9: Figuren viser de fem første trekanttallene.

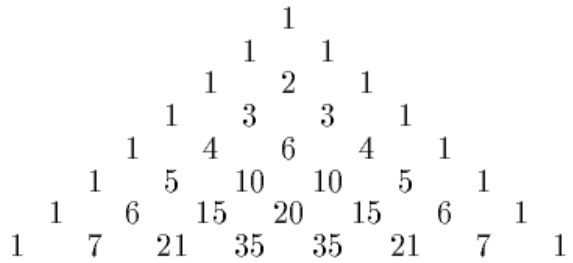
Source: Wikipedia

Vi kan derfor skrive den rekursive formelen for trekanttallet T_n gitt ved:

$$T_n = T_{n-1} + n \tag{4.8}$$

Pascals talltrekant Pascals talltrekant er et trekant utvalg av binomialkoeffisienter som brukes i sannsynlighets teori, kombinatorikk og algebra.

Hver rad i talltrekantet starter og slutter med tallet 1, og de andre tallene som ligger inn i trekanten er lik summen av de to nærmeste oppstående tallene. Figur 4.10 viser illustrasjonen av Pascals talltrekant.



Figur 4.10: Pascals talltrekant

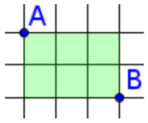
Source: Wikipedia

"Det viser seg at tallene i Pascals talltrekant forteller hvor mange ≪korteste veier≫ som leder fra toppen og fram til et krysningspunkt i talltrekanten"(Kristensen & Aanensen, 2018e).

Oppgave Gatebildet i sentrum av Kristiansand, kvadraturen, er regelmessig bygd opp med rette gater hvor gater som krysser hverandre danner vinkler på omtrent 90 grader. ≪Kvartalene≫, områdene avgrenset av gater, har tilnærmet form av rektangler.

Vi tenker oss nå byen enda mer regelmessig slik at alle ≪kvartaler≫ har en kvadratisk grunnflate.

Tenk deg at du skal gå fra gatehjørne A til gatehjørne B.

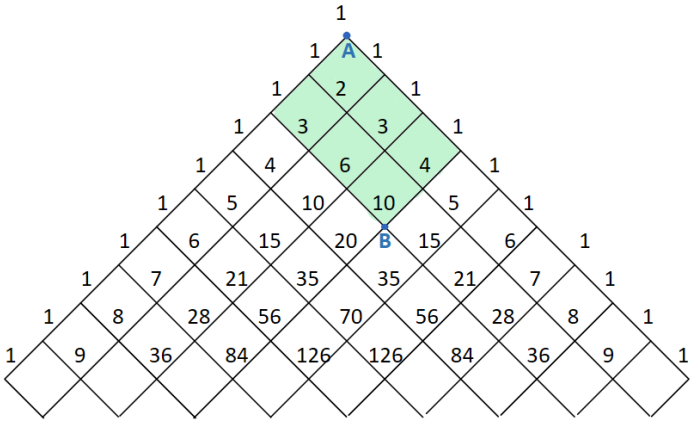


Source: NDLA

Hvor mange forskjellige «korteste veier» er det mellom A og B?

Oppgaven er hentet fra (Kristensen & Aanensen, 2018e).

Løsning Vi bruker Pascals talltrekant for å løse oppgaven. Hvis vi setter figuren gitt i oppgaven på Pascals talltrekant slik at punkt A ligger på toppunktet i trekanten, og hver node i figuren ligger på tilsvarende tallet i trekanten, vil trekanttallet som matcher punkt B vise antall mulige korteste veier.



Figur 4.11: Kvadratisk Pascals talltrekant

Som sagt hver node i Pascals talltrekant, tilsvarer en binomialkoeffisient. Derfor kan vi bruke formelen for antall permutasjoner for å finne den n-te noden i Pascals talltrekantet:

$$\binom{n}{k} = \frac{n!}{k!(n-1)!} \tag{4.9}$$

Vi kan tilpasse formelen med problemstillingen i oppgaven, og løse den. Anta n er totalt antall noder til den korteste veien fra A til B, og k er lik antall noder enten til bredden eller til lengden av rektangelet (se på den grønne rektangelet i figur 4.11) avhengig av hvilken vei man ønsker å ta.

Algoritme

1 Importere *factorial()*-funksjonen fra *math*-modulen.

2 Definere en funksjon kalt *finn_antall_korteste_veier()*; funksjonen tar n og k som parametere.

2.1 Definere en variabel *antall* for å lagre resultatet i; sette variabelen lik 0.

2.2 Bruke *try-catch*-uttrykk for å unngå eventuelle feil-verdier.

2.3 Prøve å

2.3.1 finne antall veier ved å bruke formelen 4.9. Vi benytter operatoren *heltallsdivisjon* for å få heltall som resultat.

2.4 I unntatte tilfeller med feil-verdier

2.4.1 returneres 0.

2.5 Returnere antall korteste veier.

3 Definere funksjonen *main()* som klientprogram.

3.1 Be brukeren om å oppgi tallene n og k; konvertere input-verdiene til heltall.

3.2 Finne antall korteste veier ved å kalle funksjonen *finn_antall_korteste_veier()*; funksjonen tar n og k som parametere.

3.3 Skrive ut resultatet.

4 Kalle *main()*-funksjonen.

```
'''
Programmet finner antall korteste veier fra punkt A til B vha. binomialkoeffisienter
'''

# importerer biblioteker
from math import factorial as fac
import scipy.special

'''
funksjon for å finne antall mulige korteste vei fra A til B ved hjelp av Pascals talltrekant
parametere: totalt antall noder n, antall noder til bredde eller lengde
return: antall mulige korteste vei
'''
def finn_antall_korteste_veier(n, k):

    # definere en variable antall
    antall = 0

    # bruker formelen for binomial koeffisient, try-except vil hjelpe med å unngå eventuelle feil
    try:
        antall = fac(n) // fac(k) // fac(n - k)
    except ValueError:
        antall = 0

    # kan også kalle på binom()-funksjonen fra scipy-biblioteket
    # antall = scipy.special.binom(n,k)

    # returnerer resultatet
    return antall

# klient-programmet
def main():

    # får n og k verdier fra brukeren
    n = int(input('Skriv inn totalt antall noder fra A til B: '))
    k = int(input('Hvilken vei du ønsker å ta? Skriv inn antall noder til veien: '))

    # finner antall korteste veier ved å kalle finn_antall_korteste_veier()-funksjonen
    antall = finn_antall_korteste_veier(n, k)

    # skriver ut resultatet
    print(f'Fra punkt A til B, finnes det {antall} korteste veier.')

main()
```

Program 4.10: Programmet finner antall korteste veier fra punkt A til B vha. binomialkoeffisienter.

Output:

```
Skriv inn totalt antall noder fra A til B: 5
Hvilken vei du ønsker å ta? Skriv inn antall noder til veien: 2
Fra punkt A til B, finnes det 10 korteste veier.
```

4.1.8 Eksponential vekst

Oppgave Du skal nå lage en forenklet modell for harepopulasjonen på fjellet. I et avgrenset område på fjellet settes det ut N_0 harer. Anta at populasjonsveksten er gitt ved:

$$N(t) = N_0 * 1.4^t \tag{4.10}$$

Lag en funksjon som tar inn to argumenter: antall år gitt ved t og antall harer vi starter med gitt ved N_0, og finner harepopulasjonen.

Plott deretter harepopulasjonen de første x årene. Hvorfor er modellen urealistisk?

Oppgaven er hentet fra (ProFag, 2021).

Løsning Vi først lager en funksjon *beregn_harepopulasjon()*; funksjonen tar inn de to verdiene som er bedt i oppgaven som parametere, og returnerer harepopulasjonen.

Deretter lager vi en annen funksjon for å plotte harepopulasjonen ved hjelp av denne funksjonen.

Algoritme

1 Importere *NumPy*-biblioteket, og *matplotlib.pyplot*-modulen.

2 Definere en funksjon *beregn_harepopulasjon()*; funksjonen tar antall harer ved tiden 0, og antall år som parametere.

- 2.1 Beregne harepopulasjonen N_t ved å bruke formelen 4.10.
- 2.2 Returnere N_t.

3 Definere en funksjon *plot_harepopulasjon()* for å plotte grafen.

- 3.1 Få antall harer ved tiden 0, N_0, og antall år t fra brukeren; konvertere input-verdiene til heltall.
- 3.2 Beregne harepopulasjonen N_t ved tiden t ved å kalle *beregn_harepopulasjon()*-funksjonen.
- 3.3 Skrive ut resultatet.
- 3.4 Lage en liste for x-verdier ved å kalle funksjonen *linspace()* fra *Numpy*-biblioteket; funksjonen tar startpunkt, sluttpunkt og antall punkter for plotting som parametere.
  Her setter vi startverdi lik 0 og sluttverdi lik t. Antall punkter kan være hva som helst.
- 3.5 Lage en liste for y-verdier ved å kalle *beregn_harepopulasjon()*-funksjonen, funksjonen tar startverdi N_0 og x-verdier som parametere.
- 3.6 Sette tittel for plottet ved å kalle *title()*-funksjonen fra *pyplot*.
- 3.7 Sette label for x- og y-aksen ved å kalle henholdsvis *xlabel()*- og *ylabel()*-funksjonen.
- 3.8 Sette rutenett på plottet ved å kalle *grid()*-funksjonen.

3.9 Tegne grafen ved å kalle *plot()*-funksjonen; funksjonen tar x- og y-verdier som parametere.

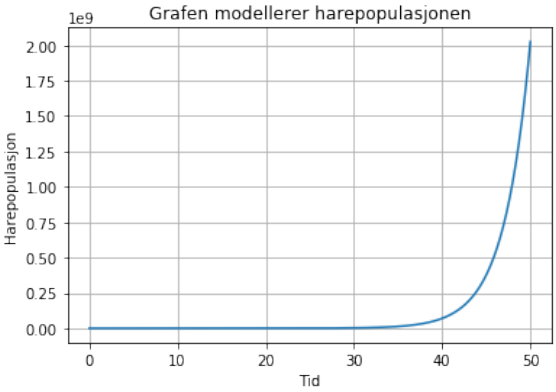
4 Kalle *plot_harepopulasjon()*-funksjonen.

```
'''
Programmet beregner harepopulasjonen N ved tiden t og tegner grafen til funksjonen
'''

# importerer biblioteker
import numpy as np
import matplotlib.pyplot as plt

'''
funksjonen beregner harepopulasjon
parameter: antall harer N_0 ved tiden 0, og antall år t
return: harepopulasjon ved tiden t
'''
def beregn_harepopulasjon(N_0, t):

    # beregner populasjon med bruk av formelen
    N_t = N_0 * 1.4**t

    # returnerer resultatet
    return N_t

# funksjonen for plotting
def plot_harepopulasjon():

    # Får startpopulasjon og antall år fra brukeren
    N_0 = int(input('Skriv inn antall harer ved tiden 0: '))
    t = int(input('Skriv inn antall år: '))

    # beregner harepopulasjonen
    N_t = beregn_harepopulasjon(N_0, t)

    # skriver ut harepopulasjonen
    print(f'Harepopulasjonen etter {t} år er {N_t}.')

    # lager en liste av x-verdier med hjelp av linspace()-funksjonen
    x_verdier = np.linspace(0, t, 1000)

    # beregner y-verdi for hver x-verdi med å kalle funksjonen beregn_harepopulasjon()
    y_verdier = beregn_harepopulasjon(N_0, x_verdier)

    # setter tittel for grafen
    plt.title("Grafen modellerer harepopulasjonen")

    # setter navn for x-aksen
    plt.xlabel("Tid")

    # setter navn for y-aksen
    plt.ylabel("Harepopulasjon")

    # lager rutenett i plottet
    plt.grid()

    # plotter grafen med å kalle plot-funksjonen() (fra pylab)
    plt.plot(x_verdier, y_verdier)

plot_harepopulasjon()
```

Program 4.11: Programmet beregner harepopulasjonen N ved tiden t og tegner grafen til funksjonen.

Eksempel output:

```
Skriv inn antall harer ved tiden 0: 100
Skriv inn antall år: 50
Harepopulasjonen etter 50 år er 2024891623.9764307.
```



Figur 4.12: Grafen viser utviklingen av harepopulasjonen som en eksponential funksjon.

"Grafen 4.12 viser en eksponential vekst av harer. Grunnen til at modellen er urealistisk er fordi den ikke tar hensyn til andre faktorer. En økende harepopulasjon ville ført til mangel på mat siden vi ser på et avlukket område. Det ville også ført til en økende populasjon av rovdyr, siden rovdyrene får mer mat. I et avlukket område ville altså harene hatt en begrenset bæreevne"(ProFag, 2021).

4.2 Ingeniørfaglige resultater

4.2.1 Mål

Målet i starten av prosjektet og slik som er definert i visjonsdokumentet Vedlegg B, var å finne ut hvilke matematiske oppgaver egner seg best for å øke dybdeforståelse, engasjement og kreativitet hos elever, og å finne anbefalte fremgangsmåter for implementasjon av disse oppgavene.

For å oppnå målet ble det gjennomført en forsking om bruk av programmering i matematikkfaget. Forskingen ble utført ved å studere en rekke vitenskapelige ressurser som hadde undersøkt bruk av programmering i matematikken.

Det viste seg at elevene som fremtidige arbeidstakere og samfunnsborgere i et stadig mer komplekst samfunn, har behov for å ha gode digitale ferdigheter og ikke bare å være brukere av digitale verktøy uten forståelse av hvordan disse verktøyene fungerer. Behovet for digitale ferdigheter hos elevene på den ene siden, og viktigheten med dybdelæring på den andre siden har ført til innføring av programmering i matematikkfaget og andre fag.

Programmering som et sterkt verktøy vil bidra til at elevene blir gode problemløsere på grunn av programmering krever at elvene tenker logisk, kritisk, og algoritmisk, noe som vil føre til dybdelæring hos elevene.

Basert på disse informasjonene ble det samlet inn en rekke matematiske oppgaver for matematikkfagene på VGS, og for disse oppgavene ble det utviklet algoritmer som viser fremgangsmåte til programmering av løsninger, og selve programkoder. For utvikling av algoritmer ble forsøkt å ta så mange detaljer som mulig slik at en matematikklærer kan skrive egne koder med bruk av algoritmene. Programkoder skal spille fasitrollen i denne sammenheng. Oppgavene er utviklet på den måten

som kan brukes som læremiddel av matematikklærerne direkte i klasserommet. Oppgaveheftet kan derfor sees både som et hjelpemiddel, men også som et læremiddel.

Blant matematikkfagene på VGS, ble det ikke tatt med yrkesfaglige matematikkfag på grunn av begrenset tid og stor mengde arbeid, men oppgavene for praktisk matematikk (P-matte) slik som kompetansemålene for disse fagene viser, har mye i felles med de yrkesfaglige matematikkfagene. Det antas dermed at oppgavene for praktisk matematikk kan være aktuelt å brukes for de yrkesfaglige matematikkfagene også.

I visjonsdokumentet er beskrevet noen retningslinjer og krav til sluttdokumentet. Her undersøkes om de definerte kravene er oppfylt, og ønskede målene er oppnådd gjennom dette prosjektet:

1 Oppgaver må velges slik at de kan bidra med dybdelæring.

   For valg av oppgaver ble det tatt hensyn til kjerneelementene i matematikkfaget, og tema som problemløsning, algoritmisk tenkning og programmering som viktige problemløsnings strategier som kan utvikle matematisk forståelse, og resultere i dybdelæring hos elevene.

2 Oppgaver må velges slik at de kan stimulere engasjement og kreativitet hos elever.

   Oppgavene ble valgt blant de oppgavene som antas til å utfordre elevene, og engasjere dem til å bruke kreativitet til å finne egne strategier for å løse oppgavene.

3 Oppgaver må lages slik at de fremviser mulige måter for algoritmisk tenkning.

   For å løse oppgavene trenger elevene å tenke algoritmisk, og finne en logisk måte for å løse oppgaven. Spesielt for problemløsnings oppgaver trenger elevene å finne først et mønster, og deretter benytte mønsteret til å skrive program.

4 Hver oppgave må inkludere i hvert fall en instruksjon som viser hvordan oppgaven kan løses (fremgangsmåten).

   Alle oppgavene inkluderer fremgangsmåten som viser skrittvis hvordan problemet kan programmeres.

5 Oppgaver må ha løsningsforslag i form av enten pseudokode, programkode eller begge deler.

   Fremgangsmåten for å løse oppgavene er på formen pseudokode (tekstlig beskrivelse av algoritmen) som beskriver skrittene for hvordan problemet kan programmeres. Det ble også lagt til programkoder for hver oppgave.

6 Løsningsforslagene bør gi output.

   For hvert program gis et eksempel output.

7 Oppgavene skal være programmerbar.

   Alle oppgavene har løsning, og kan programmeres.

4.2.2 Evaluering

Evaluering av resultater som en del av forskningsmetoden i dette prosjektet kan sees fra to sider. Først at oppgavene er i samsvar med kompetansemålene for de forskjellige matematikkfagene. Denne delen ble gjennomført samtidig som oppgavene ble valgt og samlet inn. Oppgavene er hentet fra ressurser som var nylig oppdatert og er tilgjengelige på Internett, og fra bøker som er gitt ut i de to siste årene (2019-2020). Bøkene er skrevet etter fagfornyelsen og inneholder tema som regnes som aktuelle tema i de nye læreplanene.

Etter at oppgaveheftet var ferdig ble kompetansemålene for de matematikkfagene som er inkludert i heftet, hentet fra UDIR sin nettside, og oppgavene ble sjekket mot de kompetansemålene. Deretter ble de relevante kompetansemålene satt inn i første siden av hvert kapittel slik at leserne

og de fremtidige brukerne av dokumentet kan ha klar oversikt over hva som er inneholdt i hvert kapittel.

Dokumentet burde også evalueres av matematikklærerne som testbrukere. Testen kan gjennomføres slik at det gis noen oppgaver til testere, sammen med løsnings strategi og algoritmen[7] til hver oppgave. Oppgaven til testbrukere er å ta i bruk algoritmen og skrive koden basert på instruksjoner i algoritmen. Målet med testen er å evaluere hvor godt algoritmen kan lede til programmet som er ment til å være fasit. På grunn av begrenset tid ble ikke denne delen implementert i dette prosjektet, og kan gjennomføres som videre arbeid.

4.3 Administrative resultater

For administrative resultater refereres til prosjekthåndboka[8]. Prosjekthåndboka inneholder blant annet prosjektplan, som viser fremdriftsplan og tidsforbruk for aktiviteter utført i løpet av prosjektet, møteinnkallinger og møtereferater, og timelister med statusrapport.

[7] Algoritmen viser hvordan løsningen kan programmeres.

[8] Prosjekthåndboka er ikke inkludert i rapporten etter oppdragsgiverens ønske, og leveres som et separat dokument.

5 Diskusjon

5.1 Svar til problemstillingen

I begynnelsen av rapporten ble problemstillingen definert i to deler som følger:

1 Utforske bruk av programmering i matematikkfaget, og

2 utvikle matematiske algoritmer og programkoder som samsvarer de nye læreplaner og støtter lærere i programmerings undervisning på VGS.

For å svare til problemstillingen ble det gjennomført en litteraturgjennomgang rundt forskjellige temaer som var relevant til problemet. Disse temaene er grundig forklart og drøftet i kapittel 1 og 2. Her tar vi for oss en kort oppsummering.

Det finnes mange gode grunner for innføring av programmering i matematikk, og for dagens samfunn ser behovet for å lære programmering til og med nødvendig ut.

Elevene som fremtidige samfunnsborgere og arbeidstakere har behov for å øke sine digitale ferdigheter i og med at samfunnet blir stadig mer komplekst og verden beveger seg raskt mot digitalisering. Det er viktig at elevene har evne til å tenke kritisk, logisk og algoritmisk, og kunne løse problemer på en strukturert måte. Dermed regnes programmering både som en nødvendig kompetanse i fremtiden, og som et effektivt verktøy for problemløsning. På den andre siden vil problemløsning øke dybdeforståelse hos elever, og gi elevene bedre muligheter for grundig læring.

Med fagfornyelsen og i særdeleshet innføring av programmering i skolefag utvikler elevene kompetanser som er avgjørende når de skal møte fremtidige forandringer. De vil lære å kontinuerlig tilegne seg nye kunnskap og ferdigheter som etterhvert vil være viktig for unge mennesker.

Til tross for alle positive sider ved integrering av programmering i skolefag, medfører en slik endring flere forutsetninger. Lærerkompetanse regnes i denne sammenheng som den viktigste forutsetningen. Denne forutsetningen har skapt noen utfordringer for lærerne siden de fleste av lærerne ikke har forkunnskap innen programmering. De trenger å støttes i veien til å innføre programmering i sine fag, og i dette prosjektet har vi forsøkt å svare til dette behovet.

Som svar til behovet for eksistens av hjelpemidler for å støtte lærere (i dette prosjektet matematikklærere i tjenesten), ble det utviklet et oppgaveheftet som vi kalte det ≪ProMat≫ . Oppgaveheftet består av kategoriserte matematiske oppgaver for 7 matematikkfag på VGS. Oppgavene ble valgt slik at de kan bidra med dybdelæring, og samtidig ble valgt blant de oppgavene som regnes som engasjerende, interessante, og viktige oppgaver. For at oppgavene skal bidra med dybdelæring ble forsøkt å velge de oppgavene som er i samsvar med de nye læreplanene, og blant de temaene som er introdusert som kjerneelementer i matematikkfaget.

Algoritmer som er utviklet for hver oppgave i oppgaveheftet, vil hjelpe matematikklærere til å skrive koder med å følge instruksjoner i algoritmene. Dessuten vil bruk av slike algoritmer i form av pseudokoder hjelpe elever med å lære å tenke algoritmisk før de vil begynne med å skrive kode.

Oppgaveheftet er utviklet slik at det kan brukes som et hjelpemiddel, men også som et læremiddel slik at lærerne kan bruke oppgavene som en del av sine undervisningsplaner. Dette dokumentet vil være til nytte av matematikklærere spesielt i de første årene av innføring av programmering i matematikkfaget.

5.2 Vitenskapelige resultater - Diskusjon

5.2.1 Sluttprodukt

Innsamling av oppgaver I startpunktet var tanken å samle inn oppgaver for alle matematikkfagene på VGS, men dette arbeidet viste seg etter hvert å være tidskrevende, og dermed ble ikke yrkesfaglige matematikkfag tatt med i dokumentet. Ved samling av oppgavene ble det tatt hensyn til kravene som var definert for det endelige produktet i visjonsdokumentet. Det ble også tatt hensyn til at oppgavene er i samsvar med kompetansemålene for hvert matematikkfag.

Design av oppgavehefte Ideen for å samle inn oppgavene under et separat dokument kom på hodet da det ble oppdaget at resultatet vil være stort, og det er lurt å ha et dokument som produkt i slutten av prosjektet. Dokumentet ble navngitt som oppgavehefte for matematikkfag på VGS, men vi gjør leserne oppmerksomme på at det ikke er tatt med alle temaene, og oppgavene dekker ikke alle kompetansemålene for hvert matematikkfag. Grunnen er at det finnes flere matematikkfag på VGS, og det er svært tidskrevende å lage et fullstendig kategorisert oppgavehefte for alle matematikkfagene på VGS. Det var kanskje mer effektivt å jobbe med ett fag, gjøre det ferdig og deretter ta et annet fag, og se til slutt hvor mange fag blir fullført. Likevel finnes det mulighet til å utvikle oppgaveheftet, og legge til flere andre oppgaver.

Algoritmer For hver oppgave vises fremgangsmåten for programmering av løsningen. Algoritmer er skrittvise instruksjoner på formen pseudokode, og det forventes at brukere ved å følge disse instruksjonene, kan skrive det programmet som algoritmen er skrevet for. For at dette målet oppnås, ble algoritmene utviklet så detaljert som mulig, og ble lagt til tilstrekkelig forklaring.

5.3 Ingeniørfaglige resultater - Diskusjon

5.3.1 Mål

Hensikten med prosjektet var å finne ut hvilke matematiske oppgaver egner seg best for å øke dybdeforståelse, engasjement og kreativitet hos elever. Slik som ble forklart i forrige kapitler er problemløsning en god metode for å øke dybdeforståelsen hos elevene, og algoritmisk tenkning og programmering er ment til å være to viktige problemløsnings verktøy.

Basert på disse funnene er oppgavene samlet blant de oppgavene som regnes som interessante oppgaver, og utfordrer elevene til å bruke sin kreativitet til å finne strategier for å løse oppgavene. Ved valg av oppgavene ble altså tatt hensyn til tema som problemløsning, algoritmisk tenkning, og programmering som sentrale og viktige temaer.

For hver oppgave ble det lagt til algoritmen og programkoden til løsningen. Disse algoritmene og programkodene ble utviklet mot målet for å støtte matematikklærerne i programmerings undervisning på VGS slik at de kan bruke dem i sine undervisningsplaner.

Det var også definert noen krav til det endelige dokumentet i visjonsdokumentet. Disse kravene var definert for å oppnå målet for prosjektet, og slik som ble forklart i forrige kapittelet, er disse kravene oppfylt gjennom prosjektet.

5.3.2 Evaluering

Slik som ble forklart i forrige kapittelet evaluering av sluttproduktet kan vurderes på to måter. Først å sjekke om oppgavene er relevant til kompetansemålene for tilhørende matematikkfaget. Denne delen er gjennomført i løpet av prosjektet. Den andre måten er å teste algoritmene med noen testbrukere for å sikre at algoritmene er tilstrekkelig gode, og kan brukes til å skrive programkode til løsninger. Denne delen, på grunn av begrenset tid ble ikke implementert.

5.4 Sluttprodukt: Helhetlig perspektiv

Oppgaveheftet kan tas i bruk som hjelpemiddel av matematikklærere for å lage sine undervisningsplaner for matematikkfag på VGS. Dokumentet har potensial til forbedring og videreutvikling, og etter evaluering av dokumentet, regnes med at dokumentet vill være et effektivt hjelpemiddel ved innføring av programmering i matematikkfag på VGS. Dette betyr at dokumentet kan enten brukes som et selvstendig dokument, eller som en del av et annet produkt avhengig av hva oppdragsgiveren ønsker.

Fra en samfunnsmessig perspektiv kan design og utvikling av et slikt produkt sees som en nødvendig del av prosessen for innføring av programmering i matematikkfag på VGS. Siden matematikklærere i tjenesten ikke har muligens forkunnskap i programmering, føles behovet for slike løsninger svært nødvendig for å lykkes med integrering av programmering i matematikkfag.

Matematikklærerne som ikke har hatt bakgrunn i programmering vil ha store utfordringer når de skal undervise programmering i klasserom. I dag har flere unge allerede lært programmering på egen hånd, og dette gjør arbeide med å undervise programmering uten forkunnskap enda mer krevende. Eksistens av dokumenter og produkter som oppgaveheftet i dette prosjektet vil være til nytte for lærerne, og de vil hjelpe lærerne til å føle seg mer selvsikker i klasserommet.

Oppgaven ble først begrenset i begynnelsen av prosjektet, på grunn av stor omfang av tema, fra STEM-fag til bare matematikkfag på VGS. Etter litt fremgang i prosjektet, ble igjen oppgaven begrenset til matematikkfag på VGS unntatt yrkesfaglige matematikkfag. Dette var på grunn av den begrensede tiden i en bacheloroppgave, og det var noe som omformet prosjektet underveis. Evaluering av arbeidet var en selvfølge som burde gjennomføres for å sikre arbeidets kvalitet; og det var også noe som ble påvirket av dette problemet.

I dette prosjektet fikk jeg muligheten til å styrke mine individuelle egenskaper, selv om det var av og til vanskelig å styre hele arbeidet. Det å ta hensyn til alle spekter av prosjektet, var en utfordring siden jeg jobbet alene med prosjektet. I denne forbindelsen var rollen av min veileder svært nyttig og effektiv, og hans tilbakemeldinger og veiledninger hjalp meg hele veien til å unngå avvik fra målet.

6 Konklusjon og videre arbeid

Resultatet av dette prosjektet som svar til problemstillingen viser at ved design og utvikling av produkter (dokument i dette prosjektet) som skal brukes i forbindelse med innføring av programmering i matematikkfag, bør tas hensyn til flere parametere:

- Oppgavene bør bidra med dybdeforståelse hos elevene; det er dermed viktig å tenke nøyaktig gjennom prosessen for valg av oppgaver.
- Oppgavene bør være engasjerende, interessante, og utfordrende slik at elevene stimuleres til å forsøke å finne løsning til problemet.
- Oppgavene bør velges og formuleres på den måten at elevene blir nødt til å jobbe strukturert, finne mønster, tenke algoritmisk, og konvertere tanken til løsnings strategier.
- Oppgavene bør være relevant til virkelige situasjoner slik at elevene kan imaginere problemet og få god oppfatning av det, og videre lage matematiske modeller for den virkelige situasjonen ved hjelp av sine oppfatninger.
- Programmering i skolefag (matematikk i dette prosjektet) bør benyttes som et verktøy for å stimulere elevene til å løse problemer logisk og algoritmisk, og forsterke sine digitale ferdigheter.
- Målet ved utvikling og design av produkter i forbindelse med innføring av programmering i skolefag (matematikk i dette prosjektet) er å støtte lærere, i særdeleshet lærerne i tjenesten, for å anvende programmering i sine fag. Det er derfor nødvendig å lage tilstrekkelig gode algoritmer som leder fra problem til program. Pseudokoding i denne sammenheng er et godt verktøy som viser fremgangsmåten skrittvis.
- For lærerne har programmering betydning i forhold til deres fag, og ikke ren programmering. Derfor er det viktig at oppgavene som velges er relevant til læreplaner for fag.

Evaluering av arbeidet, gjør arbeidet til å bli pålitelig, og sikrer kvaliteten. I dette prosjektet er arbeidet evaluert ved å kontrollere at oppgavene er relevant til læreplaner for ethvert matematikkfag, og som inkluderer kompetansemålene for de fagene.

Det er også behov for å evaluere dokumentet ved å teste algoritmene med noen testbrukere (matematikklærere) ved å sjekke om testbrukere kommer til å skrive koden ved hjelp av tilsvarende algoritmen. Denne delen er ikke gjennomført i dette prosjektet, og kan gjennomføres som videre arbeid.

Dokumentet kan også forbedres ved å legge til flere andre oppgaver for hvert matematikkfag. Det kan også legges til oppgaver fra matematikkfag for yrkesfaglige utdanningsprogrammer. Så kan oppgaveheftet regnes som et fullstendig hjelpemiddel for matematikklærere på VGS, og de kan bruke heftet for å lage sine egne undervisningsplaner.

Referanser

Alhefdhi, A., Dam, H. K., Hata, H. & Ghose, A. (2018). Generating pseudo-code from source code using deep learning. *2018 25th Australasian Software Engineering Conference (ASWEC)*, 21–25.

Astropy-Developers. (2021). *Table* [Link].

Astropy-Team. (udatert). *The Astropy Prosject* [Link].

Balanskat, A. & Engelhardt, K. (2015). Computing our future: Computer programming and coding - Priorities, school curricula and initiatives across Europe.

Benbasat, I. & Zmud, R. W. (1999). Empirical Research in Information Systems: The Practice of Relevance. *MIS Quarterly*, *23*(1), 3–16. http://www.jstor.org/stable/249403

Bocconi, S., Chioccariello, A. & Earp, J. (2018). The Nordic approach to introducing computational thinking and programming in compulsory education. https://doi.org/10.17471/54007

Bueie, H. (2019). *Programmering for matematikklærere*. Universitetsforlaget.

Campus-Inkrement. (udatert). *Campus Matte* [Link].

Dvergsdal, H. (2019). *Python (programmeringsspråk)* [Link].

Falcon, S. & Plaza, Á. (2007). The k-Fibonacci sequence and the Pascal 2-triangle. *Chaos, Solitons & Fractals*, *33*(1), 38–49.

Farlow, S. J. (2006). *An introduction to differential equations and their applications*. Courier Corporation.

FLIPCLASS. (udatert). *Matte 1 NTNU Slippes i mai 2021* [Link].

Forsström, S. E. & Kaufmann, O. T. (2018). A literature review exploring the use of programming in mathematics education. *International Journal of Learning, Teaching and Educational Research*, *17*(12), 18–32.

Futschek, G. (2006). Algorithmic Thinking: The Key for Understanding Computer Science. I R. T. Mittermeir (Red.), *Informatics Education – The Bridge between Using and Understanding Computers* (s. 159–168). Springer, Berlin, Heidelberg. https://doi.org/https://doi.org/10.1007/11915355

GeeksforGeeks. (2021a). *Program for Bisection Method* [Link].

GeeksforGeeks. (2021b). *Program for Newton Raphson Method* [Link].

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

Hafting, H. & Ljosland, M. (2014). *Algoritmer og datastrukturer*. Gyldendal Akademisk.

Haraldsrud, A. D., Sveinsson, H. A. & Løvold, H. H. (2020). *Programmering i skolen*. Universitetsforlaget.

Hevner, A. R. (2007). A Three Cycle View of Design Science Research. *Scandinavian Journal of Information Systems*, *19*, 87–92. https://aisel.aisnet.org/sjis/vol19/iss2/4

Hevner, A. R., March, S. T., Park, J. & Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly*, *28*(1), 75–105. http://www.jstor.org/stable/25148625

Hunter, J., Dale, D., Firing, E., Droettboom, M. & the Matplotlib development team. (2021a). *API Overview* [Link].

Hunter, J., Dale, D., Firing, E., Droettboom, M. & the Matplotlib development team. (2021b). *Matplotlib: Visualization with Python* [Link].

Hunter, J., Dale, D., Firing, E., Droettboom, M. & the Matplotlib development team. (2021c). *matplotlib.pyplot.annotate* [Link].

Hunter, J., Dale, D., Firing, E., Droettboom, M. & the Matplotlib development team. (2021d). *matplotlib.pyplot.axhline* [Link].

Hunter, J., Dale, D., Firing, E., Droettboom, M. & the Matplotlib development team. (2021e). *matplotlib.pyplot.bar* [Link].

Hunter, J., Dale, D., Firing, E., Droettboom, M. & the Matplotlib development team. (2021f). *Pyplot tutorial* [Link].

IPython-Development-Team. (udatert). *IPython: Interactive Computing* [Link].

I.T., P. (udatert). *The Euclidean division* [Link].

Jakobsen, R. E. (2019). *Programmering i matematikk– muligheter og utfordringer: En studie rundt innføringen av programmering som del av matematikkfaget i den norske skolen og hvilke argumenter som taler for eller imot innføringen.* (Masteroppgave). Universitetet i Agder. Alta.

Kafai, Y. & Burke, Q. (2013). Computer Programming Goes Back to School. *Phi Delta Kappan*, *95*, 61–65. https://doi.org/10.1177/003172171309500111

Kalvø, T., Opdahl, J. C. L., Weider, Ø. & Skrindo, K. (2020). *Mønster: Matematikk 1T*. Gyldendal Norsk Forlag.

Kaufmann, O. T. & Stenseth, B. (2020). Programming in mathematics education. *International Journal of Mathematical Education in Science and Technology*, *0*(0), 1–20. https://doi.org/10.1080/0020739X.2020.1736349

Kikora. (udatert). *Utforsking, problemløsing og trening i nye Kikora* [Link].

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B. E., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J. B., Grout, J., Corlay, S. mfl. (2016). *Jupyter Notebooks-a publishing format for reproducible computational workflows.* (Bd. 2016).

Kristensen, O. & Aanensen, S. (udatert). *Funksjoner Løsninger S2* [Link].

Kristensen, O. & Aanensen, S. (2018a). *Eksponentialfunksjon som modell* [Link].

Kristensen, O. & Aanensen, S. (2018b). *Hva er en differensiallikning?* [Link].

Kristensen, O. & Aanensen, S. (2018c). *Tallfølger* [Link].

Kristensen, O. & Aanensen, S. (2018d). *Tallrekker* [Link].

Kristensen, O. & Aanensen, S. (2018e). *Trekanttall og Pascals talltrekant* [Link].

Kristensen, O. & Aanensen, S. (2018f). *Vektorer* [Link].

Kristensen, O. & Aanensen, S. (2018g). *Økonomiske optimeringsproblem* [Link].

Kristensen, O. & Aanensen, S. (2020). *Likningssett* [Link].

Kristensen, O., Aanensen, S. & Skurdal, B. (2021). *Andre typer modeller og mønstre* [Link].

Kunnskapsdepartementet. (2004). *St.meld. nr. 30(2003-2004): Kultur for læring* [Link[Melding]].

Kunnskapsdepartementet. (2006). *Kunnskapsløftet: reformen i grunnskole og videregående opplæring* [Link].

Kunnskapsdepartementet. (2016). *Meld. St. 28 (2015–2016): Fag – Fordypning – Forståelse — En fornyelse av Kunnskapsløftet* [Link[Melding]].

Kunnskapsdepartementet. (2018a). *Fornyer innholdet i skolen* [Link[Pressemelding]].

Kunnskapsdepartementet. (2018b). *Kjerneelementer i fag* [Link[Utredning]].

Kunnskapsdepartementet. (2019). *Nye læreplaner skal gi elevene tid til mer fordypning* [Link[Pressemelding]].

Lambic, D. (2011). Presenting practical application of Mathematics by the use of programming software with easily available visual components. *Teaching Mathematics and its Applications*, *30*, 10–18. https://doi.org/10.1093/teamat/hrq014

Ludvigsenutvalget. (2014). *Elevenes læring i fremtidens skole: Et kunnskapsgrunnlag* [Link[Utredning]].

Ludvigsenutvalget. (2015). *Fremtidens skole: Fornyelse av fag og kompetanser* [Link[Utredning]].

Løvås, G. G. (2018). *Statistikk for universiteter og høyskoler*. Universitetsforlaget.

ManBearPig. (2016). *How to reverse a number mathematically* [Link].

Martinez, M. E. (1998). What is problem solving? *The Phi Delta Kappan*, *79*(8), 605–609.

MAT20x Vår 2021. (2021).

Matematikk.org. (udatert). *Arealsetningen* [Link].

NCERT. (2020). Introduction to Problem Solving. *Computer Science - Class XI* (s. 60–84). NCERT.

Nelson, J. (2018). *The Perfect Multiple Bar Chart* [Link].

NumPy. (udatert). *NumPy* [Link].

Oda, Y., Fudaba, H., Neubig, G., Hata, H., Sakti, S., Toda, T. & Nakamura, S. (2015). Learning to generate pseudo-code from source code using statistical machine translation (t). *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 574–584.

Olsen, A. L. (2005). Using Pseudocode to Teach Problem Solving. *J. Comput. Sci. Coll.*, *21*(2), 231–236.

pandas-Development-Team. (udatert). *pandas* [Link].

Pólya, G. (1957). *How to solve it: A new aspect of mathematical method*. Princeton University Press.

ProFag. (2021). *profag:vgs 20-21: Oppgaver til ProFag:VGS* [Link].

Project-Jupyter. (2021). *Jupyter* [Link].

Python-Software-Foundation. (2021a). *Built-in Functions* [Link].

Python-Software-Foundation. (2021b). *math — Mathematical functions* [Link].

Python-Software-Foundation. (2021c). *random — Generate pseudo-random numbers* [Link].

Python-Software-Foundation. (2021d). *time — Time access and conversions* [Link].

Rossen, E. (2019). *programmering (IT)* [Link].

Sanne, A., Berge, O., Bungum, B., Jørgensen, E. C., Kluge, A., Kristensen, T. E., Mørken, K. M., Svorkmo, A.-G. & Voll, L. O. (2016). *Teknologi og programmering for alle: En faggjennomgang med forslag til endringer i grunnopplæringen- august 2016* [Link[Rapport]].

Sawyer, R. K. (2006). The new science of learning. *The Cambridge handbook of the learning sciences*, *1*, 18.

SciPy-Developers. (udatert). *Scientific computing tools for Python* [Link].

Sevik, K. mfl. (2018). *Programmering i skolen* [Link].

Skrindo, K. (udatert). *Programmering i matematikk: Etterutdanningskurs for lærere i VGS* [Link].

SNL. (2020). *Kunnskapsløftet* [Link].

Stenlund, E. (2020). *Programmering og fagfornyelsen: Går erfaring ut på dato?* [Link].

Sterneckert, A. B. (2003). *Critical incident management*. CRC Press.

Storey, B. D. (udatert). *Numerical Methods for Differential Equations* [Link].

Storey, M.-A. (2006). Theories, tools and research methods in program comprehension: past, present and future. *Software Quality Journal*, *14*(3), 187–208.

SymPy-Development-Team. (udatert). *SymPy* [Link].

SymPy-Development-Team. (2021a). *Lambdify* [Link].

SymPy-Development-Team. (2021b). *Solvers* [Link].

SymPy-Development-Team. (2021c). *What is Symbolic Computation?* [Link].

The-SciPy-community. (2021a). *scipy.integrate.odeint* [Link].

The-SciPy-community. (2021b). *scipy.optimize.fsolve* [Link].

Torkildsen, S. H. (2017). *Matematisk problemløsing* [Link].

Tosi, S. (2009). *Matplotlib for Python developers*. Packt Publishing Ltd.

Tulchak, L. & Marchuk, -. (2016). *History of python* (Doktoravhandling). —-.

tutorialpoint. (udatert). *Python - Functions* [Link].

UiO. (2021). *ProFag - realfaglig programmering* [Link].

Utdanning.no. (udatert). *Videregående* [Link].

Utdanningsdirektoratet. (udatert). *Videregående opplæring* [Link].

Utdanningsdirektoratet. (2019). *Dybdelæring* [Link].

Utdanningsdirektoratet. (2020). *Hva er nytt i matematikk? Elevene skal bli gode problemløsere og forstå hvordan matematikk henger tett sammen med andre fag.* [Link].

Van Rossum, G. mfl. (2007). Python Programming Language. *USENIX annual technical conference*, *41*, 36.

vilbli.no. (udatert). *Valg av matematikk* [Link].

W3Schools. (udatert-a). *Python Classes and Objects* [Link].

W3Schools. (udatert-b). *Python Lambda* [Link].

W3Schools. (udatert-c). *Python Variables* [Link].

Walls, P. (2019a). *First Order Equations* [Link].

Walls, P. (2019b). *Riemann Sums* [Link].

Wangen, C. (2020). *Dybdelæring i matematikk: En empirisk studie om dybdelæring i matematikk knyttet til Fagfornyelsen* (Masteroppgave). Universitetet i Oslo. Oslo.

Weisstein, E. W. (2002a). Integral. *https://mathworld. wolfram. com/.*

Weisstein, E. W. (2002b). Palindromic Number. *https://mathworld. wolfram. com/.*

Weisstein, E. W. (2002c). Square number. *https://mathworld. wolfram. com/.*

Weisstein, E. W. (2002d). Triangular number. *https://mathworld. wolfram. com/.*

Young, J. (2020). *Frequency Distribution* [Link].

Vedlegg

Vedlegg inneholder:

A Oppgavehefte

B Visjonsdokument

Det ble valgt å ikke inkludere kravdokumentasjon, siden prosjektet ikke er av type systemutviklings prosjekter; men det er definert noen krav til det endelige dokumentet i visjonsdokumentet, Vedlegg B. Prosjektet vil heller ikke inkludere systemdokumentasjon for samme grunn.

A Oppgaveheftet

1 Introduksjon

I dette dokumentet har vi forsøkt å samle inn oppgaver fra matematikkfag på videregående skole. Vi har satset på å tilpasse oppgavene med nye læreplaner etter fagfornyelsen. Dermed har fokuset vært mest på tema som algoritmisk tenkning, problemløsning, modellering, og analyse av data ved hjelp av programmering, og mer spesifikt Python programmering. Dokumentet begynnes med en kort introduksjon til programmering i Python. Videre består dokumentet av følgende kapitler:

- Matematikk 1P: Praktisk matematikk på Vg1.
- Matematikk 1T: Teoretisk matematikk på Vg1.
- Matematikk 2P: Praktisk matematikk på Vg2.
- Matematikk R1: Matematikk for realfag på Vg2.
- Matematikk R2: Matematikk for realfag på Vg3.
- Matematikk S1: Matematikk for samfunnsfag på Vg2.
- Matematikk S2: Matematikk for samfunnsfag på Vg3.

I hvert kapittel tar vi for oss noen tema innen faget som underkapitler. Disse delene for det meste, består av en kort teoridel, noen oppgaver relatert til temaet, utdypning av oppgavene, og strategier og algoritmer for programmering av løsninger. Det har også blitt lagt til kildekode og output av løsningsforslaget for hver oppgave.

Til slutt håper vi at dett dokumentet vil være en liten hjelp for matematikklærere til å anvende programmering i matematikk. Dokumentet dekker selvfølgelig ikke alle temaene og kompetansemålene for de nevnte fagene, men kan uansett benyttes som eksempel i andre temaene også.

2 Programmering i Python

Vi antar at lesere av dette dokumentet er allerede kjent med Python, og vet om grunnleggende begreper og funksjoner i dette programmeringsspråket. I tilfellet leseren er en nybegynner, anbefales å lese Python Tutorial for litt forberedelse. Det finnes også flere gode nettsider som W3Schools som tilbyr in-browser koding for de som vil lære Python. Her nevnes noen av de grunnleggende begrepene og funksjonene, men vi går ikke så dypt i forklaringer.

2.1 Variabler

Variabler brukes for lagring av verdier. Det finnes ikke noen kommandoer for deklarering av variabler i Python; Variabler blir til når vi setter verdi for dem (W3Schools, udatert-c). Når vi oppretter en variabel i en funksjon, kalles variabelen, en *lokal variabel*, og kan bare brukes i den funksjonen. Variabler som lages utenfor en funksjon er kjent som *globale variabler*. Globale variabler kan brukes både inn i funksjoner og utenfor.

2.2 Datatyper

Variabler kan lagre data av forskjellige typer. De viktigste datatyper i Python er:

- Strenger: str
- Tall:
  - int →heltall
  - float →flyttall/desimaltall (desimaltall kan også defineres med å importere *decimal*-biblioteket. Forskjellen mellom decimal.Decimal og float er i deres rekkevidde)
  - complex →komplekse tall
- Sekvenstyper: liste, tupel, range
- Mapping-type: dict (dictionary)
- Settyper: set, forzenset
- Boolsk type: bool
- Binære typer: byte, bytearray, memoryview

2.3 Operatorer

Operatorer deles i forskjellige grupper:

- Regneoperatorer: Tabellen A.1 viser de regneoperatorene som brukes oftest i matematisk programmering.

Operasjon	Operator
Addisjon	+
Subtraksjon	-
Multiplikasjon	*
Divisjon	/
Potens	**
Kvadratrot	**(1/2)
Nte-rot	**(1/n)
Heltallsdivisjon	//
Restdivisjon	%
Parentes	()

Tabell A.1: Regneoperatorer i Python

- Assignment-operatorer: I Python brukes = for tilordning mellom variabler og verdier, for eksempel:

```
x = 5
x, y, z = 1, -1, 1
navn = 'Sara'
```

  Det finnes også noen sammensatte operatorer i Python som +=, −=, ∗=, /=, %=, //=, **= , &=, |=, ∧=, >>=, <<=. For eksempel $x \mathrel{+}= 5$ legger 5 til variabelen x og lagrer den nye verdien i den; uttrykket tilsvarer altså $x = x + 5$.

- Sammenlignings operatorer: Tabellen A.2 viser disse operatorene.

Matematikk	Python	Beskrivelse
$<$	$<$	Mindre enn
$>$	$>$	Større enn
$=$	$==$	Lik
$\leq$	$\leq$	Mindre enn eller lik
$\geq$	$\geq$	Større enn eller lik
$\neq$	!=	Ulik

Tabell A.2: Sammenlignings operatorer i Python

2.4 Regnerekkefølgen

Regnerekkefølgen har betydning i Python, tabellen A.3 lister operatorprioriteten fra høyeste til laveste. Operatorer i samme rad har samme prioritet.

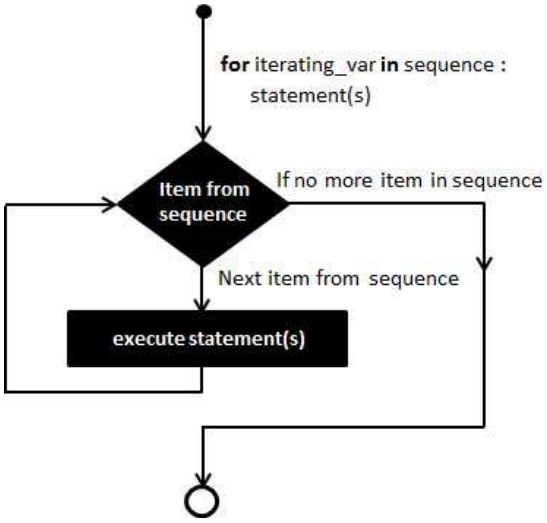
Operator	Beskrivelse
()	Parenteser
**	Potenser
*, /, //, %	Multiplikasjon, Divisjon, Heltallsdivisjon, Restdivisjon
+, -	Addisjon, Subtraksjon

Tabell A.3: Regnerekkefølge i Python

2.5 Løkker

Python har to typer løkker:

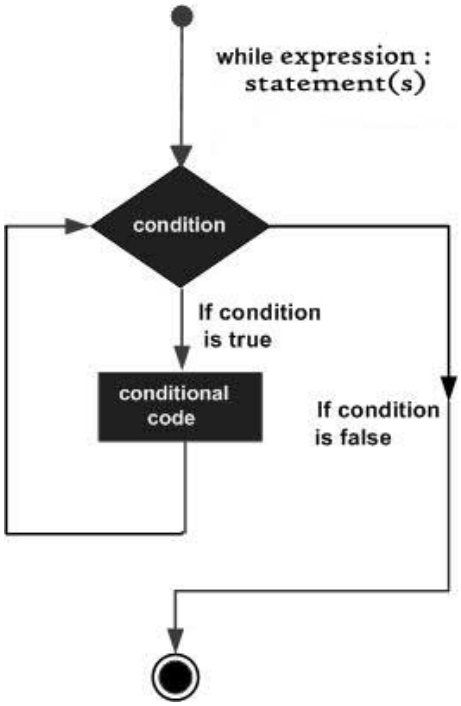
- For-løkke: I Python går for-løkken over elementene i en sekvens som kan være en liste eller en streng, i den rekkefølgen de vises i sekvensen.



Figur A.1: Flytdiagram for for-løkke

Source: tutorialspoint

- While-løkke: I en while-løkke utfører vi en rekke instruksjoner så lenge en betingelse er sant.



Figur A.2: Flytdiagram for while-løkke

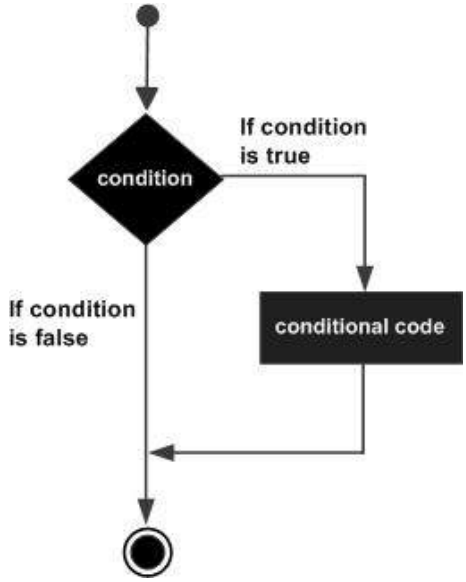
Source: tutorialspoint

- Nestet while-løkke: Når vi har flere while-løkker inn i hverandre, kalles det en nestet while-løkke.

2.6 Betingelser

I programmering er det ofte behov for programmer der vi tar en eller annen beslutning som følge av en betingelse. I Python benyttes følgende setninger når vi skal programmere slike situasjoner:

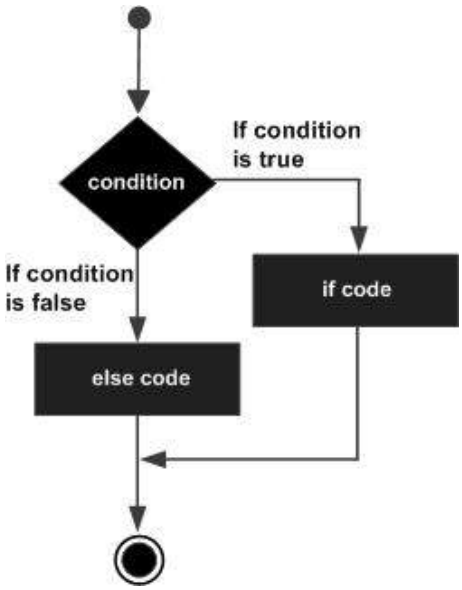
- If-setning



Figur A.3: Flytdiagram for if-setning

Source: tutorialspoint

- If-else-setning



Figur A.4: Flytdiagram for if-else-setning

Source: tutorialspoint

- Nestet if-setning (if-elif-elif-...-else): Når vi har flere betingelser, bruker vi nestet if-setning.

2.7 Built-in funksjoner

Python-interpreter har en rekke funksjoner innebygd i den som alltid er tilgjengelige (Python-Software-Foundation, 2021a). Tabell A.4 viser de av dem som antas å brukes mest i dette dokumentet.

Funksjon	Beskrivelse
abs()	Returnerer absoluttverdi av et tall
float()	Returnerer et flyttall
input()	For å få input fra bruker
int()	Returnerer et heltall
len()	Returnerer lengde av et objekt
list()	Returnerer en liste
Print()	Skriver ut data
range()	Returnerer en sekvens av tall, som standard starter fra 0 og inkrementerer tallene med 1
round()	Returnerer rundverdi av et tall
str()	Returnerer et string objekt

Tabell A.4: Noen eksempler av Python built-in funksjoner

2.8 Egendefinerte funksjoner

Det er også mulig å definere egne funksjoner i Python som i alle andre programmeringsspråk. En funksjon er en blokk av koder som kan utføre en handling. Fordelen med funksjoner er at de er gjenbrukbare, og gir bedre modularitet og den nødvendige funksjonaliteten til programmet (tutorialpoint, udatert).

Følgende regler gjelder ved definering av en funksjon i Python (tutorialpoint, udatert):

1 Funksjoner defineres med nøkkelordet «def», navn til funksjonen og parenteser.

2 Det er mulig å legge til parameter(e) inn i parentesene.

3 Etter på kan vi skrive en eller flere linjer av instruksjoner som vi ønsker å utføres via funksjonen.

4 Hver blokk for en funksjon startes med en kolon (:), og neste linjene er innrykket.

5 Hvis vi ønsker at funksjonen returnerer noe, bruker vi «return»-uttrykket etterfulgt av et uttrykk.

Syntaks:

```
def funksjonsnavn(parameter):
    en linje med instruksjon
    ...
    return [uttrykk]
```

3 Matematikk 1P

Kompetansemål etter matematikk 1P

Mål for opplæringa er at eleven skal kunne

- bruke prosent, prosentpoeng, promille og vekstfaktor i utrekningar og presentere og grunngi løysingar
- tolke og bruke samansette måleiningar i praktiske samanhengar og velje eigna måleining
- tolke og bruke funksjonar i matematisk modellering og problemløysing
- bruke digitale verktøy i utforsking og problemløysing knytt til eigenskapar ved funksjonar, og diskutere løysingane

Kilde

3.1 Tallregning

Oppgave Lag et program som tar et antall sekunder som input. Programmet skal ta tallet, og regne det om til dager, timer, minutter og sekunder. Til slutt skal du skrive ut hvor mange dager, timer, minutter og sekunder det ble.

Prøv å lag koden slik at kun det som er relevant, blir oppgitt. For eksempel, hvis det bare er 34 minutter og 10 sekunder, trenger vi ikke få oppgitt at det er 0 dager og timer.

Oppgaven er hentet fra (ProFag, 2021).

Løsning For å løse oppgaven trenger vi følgende informasjon:

- ett *minutt* er 60 sekunder
- én *time* er 60 minutter
- én *dag* er 24 timer

Disse verdiene definerer vi som konstanter. Det å definere slike faste variabler som konstant, hjelper oss å skrive mer oversiktlige koder.

Neste steg er å finne hvor mange minutter som finnes i det antallet sekunder, så finne hvor mange timer som finnes i det antallet minutter, og til slutt finne hvor mange dager som finnes i det antallet timer.

For å finne for eksempel antall minutter, benytter vi *heltallsdivisjon*-operatoren i Python. Og får å finne de gjenværende sekunder, benytter vi *restdivisjon*-operatoren.

Videre skal vi skrive ut bare ikke-null verdier som resultat, for dette trenger vi sette betingelser med bruk av $if - elif - else$-setninger.

Algoritme

1 Definere konstanter for antall sekunder per minutt, antall minutter per time, og antall timer per dag.

2 Få antall sekunder fra brukeren ved å kalle *input()*-funksjonen, og konvertere verdien til heltall.

3 Finne antall **minutter** ved å bruke heltallsdivisjon.

4 Finne antall sekunder vi har igjen ved å bruke restdivisjon.

5 Finne antall **timer** ved å bruke heltallsdivisjon.

6 Finne antall minutter vi har igjen ved å bruke restdivisjon.

7 Finne antall **dager** ved å bruke heltallsdivisjon.

8 Finne antall timer vi har igjen ved å bruke restdivisjon.

9 Hvis både antall dager, timer og minutter er lik null,

  9.1 skrive ut bare antall sekunder.

10 Eller hvis både antall dager og timer er lik null,

  10.1 skrive ut antall minutter og sekunder.

11 Eller hvis bare antall dager er lik null,

11.1 skrive ut antall timer, minutter og sekunder.

12 Ellers

12.1 skrive ut alle verdiene.

```
'''
Programmet konverterer antall sekunder til dag, timer, minutter og sekunder
'''

# definerer konstanter
sekund_minutt = 60
minutt_time = 60
time_dag = 24

# får input fra brukeren
antall_sekunder = int(input("Hvor mange sekunder ønsker du å konvertere? Skriv et heltall: "))

# bruker heltallsdivisjon for å finne antall minutter
minutter = antall_sekunder // sekund_minutt

# bruker restdivisjon for å finne antall sekunder vi har igjen
sekunder = antall_sekunder % sekund_minutt

# bruker heltallsdivisjon for å finne antall timer
timer = minutter // minutt_time

# bruker restdivisjon for å finne antall minutter vi har igjen
minutter = minutter % minutt_time

# bruker heltallsdivisjon for å finne antall dager
dager = timer // time_dag

# bruker restdivisjon for å finne antall timer vi har igjen
timer = timer % time_dag

# sjekker om både minutter, timer og dager er null
if (minutter==0 and timer==0 and dager==0):
    print(f'{antall_sekunder} sekunder er {sekunder} sekunder')

# sjekker om både timer og dager er null
elif (timer==0 and dager==0):
    print(f'{antall_sekunder} sekunder er {minutter} minutter og {sekunder} sekunder')

# sjekker om dager er null
elif dager==0:
    print(f'{antall_sekunder} sekunder er {timer} timer, {minutter} minutter og {sekunder} sekunder')

# ellers har alle variablene en ikke-null verdi
else:
    print(f'{antall_sekunder} sekunder er {dager}dager,{timer}timer,{minutter}minutter og {sekunder}sekunder')
```

Program A.1: Programmet konverterer antall sekunder til dag, timer, minutter og sekunder.

Eksempel output:

```
Hvor mange sekunder ønsker du å konvertere? Skriv et heltall: 57834
57834 sekunder er 16 timer, 3 minutter og 54 sekunder
```

3.2 Prosentregning

Oppgave Lag et program som spør hvor mange meter over havet noen befinner seg, og deretter printer "n er m% av høyden til Mount Everest". for eksempel: "345.60 m er 3.91 prosent av Mount Everest"

Oppgaven er hentet fra (ProFag, 2021).

Løsning Høyden til Mount Everest er 8848 meter. Vi finner prosentandelen ved følgende formel:

$$Prosentandel = \frac{hoyde}{mount\_everest\_hoyde} * 100 \tag{A.1}$$

Algoritme

1 Definere høyden til Mount Everest som konstant.

2 Få høyde-verdien fra brukeren ved hjelp av *input()*-funksjonen; og konvertere den til flyttall.

3 Beregne prosentandelen ved å bruke formelen A.1.

4 Avrunde prosentandels verdien til to desimaler nøyaktighet ved å kalle *round()*-funksjonen. Dette er valgfri; grunnen for avrunding er å ha mer oversiktlig resultat.

5 Skrive ut resultatet.

```
'''
Programmet finner prosentandelen ift. høyden for Mount Everest
'''

# definerer konstanter
mount_everest_hoyde = 8848

# får høyde i meter fra brukeren, og konverterer input-verdien til float
hoyde = float(input("Hvor mange meter over havet befinner du deg? "))

# regner ut prosentandelen
prosentandel = (hoyde/mount_everest_hoyde)*100

# runder av prosentandelen med 2 desimal nøyaktighet
prosentandel = round(prosentandel,2)

# skriver ut resultatet
print(f'Du er på {prosentandel}% av Mount Everest ved {hoyde} meter over havet')
```

Program A.2: Programmet finner prosentandelen til en gitt posisjon over havet ift. høyden for Mount Everest.

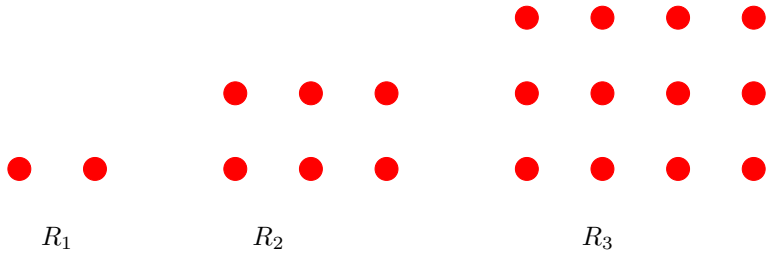
Eksempel output:

```
Hvor mange meter over havet befinner du deg? 10
Du er på 0.11% av Mount Everest ved 10.0 meter over havet
```

3.3 Matematisk modellering og problemløsning

3.3.1 Rektangeltall

Oppgave Rektangeltallene kan framstilles slik figuren viser:



Vi kaller det første rektangeltallet R_1, det neste rektangeltallet kaller vi R_2, det tredje rektangeltallet kaller vi R_3 og så videre.

Finn en matematisk modell som beskriver antall prikker i rektangeltallene. La x være nummeret på rektangeltallet, og $P(x)$ være antall prikker i tallet, plott funksjonen $P(x)$.

Oppgaven er hentet fra (Kristensen mfl., 2021).

Løsning For å lage hvert neste rektangel, legger vi til en kolonne og en rad til det forrige rektangelet. Vi kan betrakte her to mønstre:

- Mønster 1:

$$\begin{aligned}
R_1 &= 0+1=1 \quad \times \quad 1+1=2 \implies 2 \\
R_2 &= 1+1=2 \quad \times \quad 2+1=3 \implies 6 \\
R_3 &= 2+1=3 \quad \times \quad 3+1=4 \implies 12 \\
&\vdots \\
R_n &= (n-1)+1 \quad \times \quad n+1 \implies n(n+1)=n^2+n
\end{aligned}$$

- Mønster 2:

$$\begin{aligned}
R_1 &= 1\times 1+1=2 \\
R_2 &= 2\times 2+2=6 \\
R_3 &= 3\times 3+3=12 \\
&\vdots \\
R_n &= n\times n+n \to \quad n^2+n
\end{aligned}$$

Altså:

$$P(x)=x^2+x$$

Vi kan nå plotte funksjonen $P(x)$.

Algoritme

1 Importere *matplotlib.pyplot*-modulen og *Numpy*-biblioteket.

2 Definere en funksjon som returnerer $P(x)$. Funksjonen tar x som parameter.

3 Definere en funksjon *plott_funksjon()* for å plotte $P(x)$.

- 3.1 Definere en liste av x-verdier ved hjelp av *linspace()*-funksjonen fra *Numpy*-biblioteket; funksjonen tar startpunkt, sluttpunkt og antall punkter som parametere.
- 3.2 Beregne y-verdier ved å kalle *P()*-funksjonen; funksjonen tar x-verdier som parameter.
- 3.3 Sette tittel for plottet ved å kalle *title()*-funksjonen fra *matplotlib.pyplot*-modulen.

3.4 Sette x- og y-label for plottet ved å kalle henholdsvis *xlabel()*- og *ylabel()*-funksjonen fra *matplotlib.pyplot*-modulen.

3.5 Sette rutenett for plottet ved å kalle *grid()*-funksjonen fra *matplotlib.pyplot*-modulen.

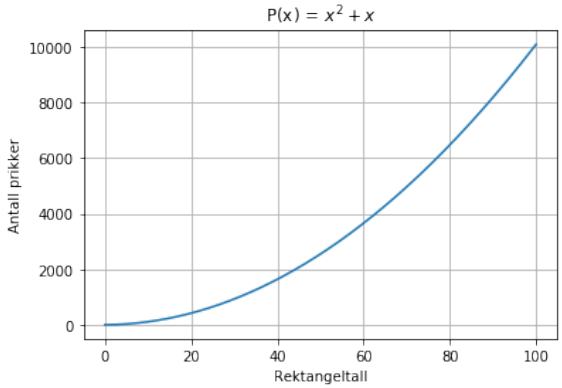
3.6 Plotte funksjonen ved å kalle *plot()*-funksjonen fra *matplotlib.pyplot*-modulen.

4 Kalle *plott_funksjon()*.

```
'''
Programmet finner antall prikker for et rektangeltall R_n
'''

# importerer biblioteker/moduler
import numpy as np
import matplotlib.pyplot as plt

# definere funksjonen P(x)
def P(x):
    return x**2+x

# funksjon for å plotte P(x)
def plott_funksjon():

    # definere x_verdier
    x_verdier = np.linspace(0, 100, 1000)

    # beregne y_verdier med å kalle funksjonen P(x)
    y_verdier = P(x_verdier)

    # sette title, x-label og y-label
    plt.title('P(x) = $x^2 + x$')
    plt.xlabel('Rektangeltall')
    plt.ylabel('Antall prikker')

    # setter rutenett for plottet
    plt.grid()

    # plotter funksjonen
    plt.plot(x_verdier, y_verdier)

plott_funksjon()
```

Program A.3: Programmet finner antall prikker for et rektangeltall R_n.

Output:



Figur A.5: Grafen viser antall prikker for et rektangeltall R_n.

Vi kan lese antall prikker for et rektangeltall R_n fra grafen A.5.

3.3.2 Funksjoner og modellering

Oppgave

a Lag en funksjon som regner ut kokepunktet til vann i en gitt høyde over havet: Vi velger en forenklet modell:

$$T_c = 100 - 0.0032h \tag{A.2}$$

der h oppgis i meter, og temperaturen kommer ut i celsius.

Sjekk at funksjonen returnerer riktig verdi ved havnivå (100 C) og på toppen av Mount Everest (ca. 72 C). Mount Everest er 8848 m høyt.

b Vi skal nå koke egg. Tiden t i minutter det tar å oppnå en plommetemperatur på T_{plomme} kan modelleres som:

$$t = A * ln[\frac{2(T_{vann} - T_0)}{T_{vann} - T_{plomme}}] \tag{A.3}$$

Der A er en konstant som kommer an på egget, T_{vann} er vanntemperaturen i kjelen (typisk kokepunktet), T_0 er temperaturen i egget før det går i gryta og T_{plomme} er temperaturen vi ønsker i plommen. For et vanlig egg er $A = 3.75$ minutter en fornuftig verdi. Lag en funksjon som implementerer denne modellen, og sjekk at den gir rimelige resultater for bløtkokt (65 C) og hardkokt (85 C) egg.

Koketid for egg er typisk 4-7 minutter for bløtkokt, og 7-10 minutter for hardkokt.

c Endre funksjonen fra del *b* slik at den tar inn høyde over havet og bruker funksjonen fra del *a* til å regne ut temperaturen i vannet.

Hva skjer om man ber om et hardkokt egg på Mount Everest? Kan du endre programmet slik at det oppfører seg penere?

d Plott tiden det tar å koke egg som funksjon av hvor varm man ønsker plommen. Plott flere linjer, der hver linje representerer en gitt høyde over havet.

Oppgaven er hentet fra (ProFag, 2021).

Løsning

a Vi lager en funksjon *finn_kokepunkt()*, som tar høyde h som parameter og returnerer kokepunkt T_c. Deretter tester vi metoden for $h = 0$ dvs. ved havnivå, og for $h = 8848$ som er høyden til Mount Everest.

Algoritme

1 Definere en funksjon *finn_kokepunkt()*.
   1.1 Beregne kokepunkt ved å bruke formelen A.2.
   1.2 Returnere kokepunktet.
2 Definere funksjon *main()* som klientprogram.
   2.1 Få høyde fra brukeren ved å kalle *input()*-funksjonen, og konvertere den til flyttall.
   2.2 Beregne kokepunkt ved å kalle *finn_kokepunkt()*-funksjonen.
   2.3 Skrive ut resultatet.

3 Kalle *main()*-funksjonen.

```python
# a: programmet finner kokepunkt til vann ved høyden h over havet

'''
funksjonen finner kokepunkt for vann ved høyden h
parameter: høyde h
return: kokepunkt T_c
'''
def finn_kokepunkt(h):

    # beregner kokepunkt med å bruke formelen som er oppgitt
    T_c = 100 - 0.0032 * h

    # returnerer resultatet
    return T_c

# klient-metoden
def main():

    # får høyde-verdien fra brukeren, konverterer den til float
    h = float(input('Skriv inn høyden du vil finne kokepunkt på: '))

    # finner kokepunktet ved å kalle finn_kokepunkt()-funksjonen
    T_c = finn_kokepunkt(h)

    # Skrive ut resultatet
    print(f'Kokepunkt ved høyden {h} m er {T_c} celsius.')

main()
```

Program A.4: Programmet finner kokepunkt til vann ved høyden h over havet.

Tester metoden for kokepunkt til vann ved havnivå:

```
Skriv inn høyden du vil finne kokepunkt på: 0
Kokepunkt ved høyden 0.0 m er 100.0 celsius.
```

Tester metoden for kokepunkt til vann på toppen av Mount Everest:

```
Skriv inn høyden du vil finne kokepunkt på: 8848
Kokepunkt ved høyden 8848.0 m er 71.69 celsius.
```

b Vi definerer en funksjon *beregn_koketid()* som tar T_0 og T_{plomme} som parametere. Så definerer vi A og T_{vann} som konstanter, og deretter beregner vi koketiden ved å bruke formelen gitt i oppgaven.

Vi lager videre en funksjon for å løse oppgaven, dvs. å finne koketid for et bløtkokt egg, og hardkokt egg.

Algoritme

1 Importere biblioteket *Numpy*.

2 Definere en funksjon kalt *beregn_koketid()*.

2.1 Definere konstanter A og T_{vann}.

2.2 Beregne koketiden ved å bruke formelen A.3.

2.3 Returnere koketiden.

3 Definere *main()*-funksjon som klientprogram.

3.1 Få T_0 og T_{plomme} som input fra brukeren; konvertere input-verdiene til flyttall.

3.2 Beregne koketiden ved å kalle *beregn_koketid()*-funksjonen.

3.3 Skrive ut resultatet.

4 Kalle *main()*-funksjonen.

```python
# b: programmet implementerer modellen for koketiden til egg

# importere numpy-biblioteket
import numpy as np

'''
funksjonen beregner koketid til egg
parametere: T_vann, T_plomme, T_0
return: tid
'''
def beregn_koketid(T_0, T_plomme):

    # definerer konstanter
    A = 3.75
    T_vann = 100

    # beregner tiden med bruk av formelen
    tid = A * np.log((2*(T_vann - T_0)) / (T_vann - T_plomme))

    # returnerer resultatet
    return tid

def main():
    T_0 = float(input('Oppgi temperaturen til egget ved tiden 0: '))
    T_plomme = float(input('Oppgi temperaturen du ønsker i plommen: '))

    #Kaller på funksjonen for å finne koketid
    koketid = beregn_koketid(T_0, T_plomme)

    # Skriver ut resultatet
    print(f'Koketid for egget er {koketid:.2f} minutter')

main()
```

Program A.5: Programmet implementerer modellen for koketiden til egg.

Tiden for et bløtkokt egg:

```
Oppgi temperaturen til egget ved tiden 0: 5
Oppgi temperaturen du ønsker i plommen: 65
Koketid for egget er 6.34 minutter
```

Tiden for et hardkokt egg:

```
Oppgi temperaturen til egget ved tiden 0: 5
Oppgi temperaturen du ønsker i plommen: 85
Koketid for egget er 9.52 minutter
```

c Vi modifiserer *beregn_koketid()*-funksjonen ved å legge til et parameter h for høyde.

Vanntemperaturen T_{vann} (som var konstant i tidligere deloppgaven) finner vi nå ved å kalle *finn_kokepunkt()*-funksjonen.

Algoritmen for programmet er altså nesten den samme som forrige deloppgave.

Algoritme

1 Importere biblioteket *Numpy*.

2 Definere en ny funksjon for å beregne koketid; funksjonen tar høyde h, T_0 og T_{plomme} som parametere.

- 2.1 Definere konstanten A.
- 2.2 Definere en variabel for koketiden, her kaller vi variabelen *tid*.
- 2.3 Finne vanntemperaturen for eggkoking ved å kalle funksjonen *finn_kokepunkt()*.
- 2.4 Bruke *try-except*-uttrykk for å håndtere eventuelle feil [9]. (Eventuelt kan vi bruke *if-else*-setning for å sette betingelse om *nevneren* <= 0).
- 2.5 Prøve å
  - 2.5.1 beregne koketiden ved å bruke formelen A.3.
- 2.6 I unntatte tilfeller
  - 2.6.1 Skrive ut en melding for brukeren at programmet feiler.
- 2.7 Returnere koketiden.

3 Definere *main()*-funksjon som klientprogram.

- 3.1 Få høyden h, starttemperaturen T_0 og plommetemperaturen T_{plomme} som input fra brukeren.
- 3.2 Beregne koketiden ved å kalle *beregn_koketid_gitt_høyde()*-funksjonen. (Funksjonen kan selvfølgelig kalles hva som helst.)
- 3.3 Skrive ut resultatet.

4 Kalle *main()*-funksjonen.

[9] Dette er fordi oppgaven ber om å finne kokepunkt for et hardkokt egg på Mount Everest der kokepunkt er 72 celsius mens temperaturen for et hardkokt egg er på ca. 85 celsius. Dette fører til at vi får negativ verdi i nevneren og programmet vil gi feil beskjed.

```python
# c: programmet beregner koketid til egg ved gitt vanntemp.

# importere numpy-biblioteket
import numpy as np

'''
funksjonen beregner vanntemperaturen i høyde h, og koketid til egg
parametere: h, T_plomme, T_0
return: tid
'''
def beregn_koketid_gitt_høyde(h, T_0, T_plomme):

    # definerer konstant A
    A = 3.75

    # definerer en variable tid
    tid = 0

    # finner vanntemperaturen med å kalle finn_kokepunkt()
    T_vann = finn_kokepunkt(h)

    # bruker try-except for å håndtere eventuelle feil
    try:
        # beregner koketiden med bruk av formelen
        tid = A * np.log((2*(T_vann - T_0)) / (T_vann - T_plomme))

    except:
        print('Noe gikk galt!')


    '''
    eventuelt kan vi bruke if-else-setning for å sette betingelse om nevneren <= 0:

    if (T_vann - T_plomme) <= 0:
        print('Nevneren er mindre enn eller lik 0')

    else:
        tid = A * np.log((2*(T_vann - T_0)) / (T_vann - T_plomme))
    '''

    # returnerer resultatet
    return tid

def main():

    # får høyde-verdien fra brukeren, konverterer den til float
    h = float(input('Skriv inn høyden du vil finne kokepunkt på: '))

    T_0 = float(input('Oppgi temperaturen til egget ved tiden 0: '))
    T_plomme = float(input('Oppgi temperaturen du ønsker i plommen: '))

    #Kaller på funksjonen for å finne koketid
    koketid = beregn_koketid_gitt_høyde(h, T_0, T_plomme)

    # Skriver ut resultatet
    print(f'Koketid for egget er {koketid:.2f} minutter')

main()
```

Program A.6: Programmet beregner koketid til egg ved gitt vanntemperatur.

Output:

```
Skriv inn høyden du vil finne kokepunkt på: 8848
Oppgi temperaturen til egget ved tiden 0: 5
Oppgi temperaturen du ønsker i plommen: 85
Koketid for egget er nan minutter
```

Vi ser at det ikke er mulig å ha et hardkokt egg på Mount Everest!

d Vi skal i denne deloppgaven bruke *matplotlib.pyplot*-modulen sammen med *Numpy*-biblioteket for å plotte koketiden mot plommetemperaturen.

For å ha flere linjer beregner vi koketiden ved forskjellige høyder, dvs. vi lager flere sett av y-verdier.

Algoritme

1 Importere modulen *matplotlib.pyplot*, og biblioteket *Numpy*.

2 Lage en liste av x-verdier ved hjelp av *linspace()*-funksjonen. Funksjonen tar startpunkt, sluttpunkt og antall punkter for plotting som parametere.

Vi setter start- og sluttpunkt lik henholdsvis 0 og 100, antall punkter velger vi å sette lik 1000 punkter.

3 Definere flere sett av y-verdier ved å kalle funksjonen *beregn_koketid_gitt_høyde()*. Vi setter forskjellige høyder h, konstant starttemperatur lik 5, og x-verdier som parametere.

4 Sette tittel for plottet ved å kalle *title()*-funksjonen fra *matplotlib.pyplot*-modulen.

5 Sette x- og y-label for plottet ved å kalle henholdsvis *xlabel()*- og *ylabel()*-funksjonen fra *matplotlib.pyplot*-modulen.

6 Sette rutenett for plottet ved å kalle *grid()*-funksjonen fra *matplotlib.pyplot*-modulen.

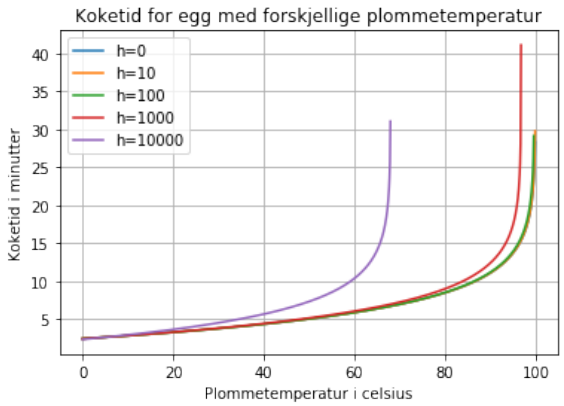
7 Plotte grafene ved å kalle *plot()*-funksjonen fra *matplotlib.pyplot*-modulen. Funksjonen tar x- og y-verdier som parametere.

8 Sette label for grafene ved å kalle *legend()*-funksjonen.

```python
# d: programmet plotter koketiden for egg avhengig av plommetemp.

# importerer moduler/biblioteker
import matplotlib.pyplot as plt
import numpy as np

# setter start- og sluttpunkt lik koketid for et bløtkokt og hardkokt egg
x_verdier = np.linspace(0, 100, 1000)

# lager 4 sett y_verdier ved h=0, h=10, h=100, h=1000
y_verdier_0 = beregn_koketid_gitt_høyde(0, 5, x_verdier)
y_verdier_10 = beregn_koketid_gitt_høyde(10, 5, x_verdier)
y_verdier_100 = beregn_koketid_gitt_høyde(100, 5, x_verdier)
y_verdier_1000 = beregn_koketid_gitt_høyde(1000, 5, x_verdier)
y_verdier_10000 = beregn_koketid_gitt_høyde(10000, 5, x_verdier)

# setter tittel for grafen
plt.title('Koketid for egg med forskjellige plommetemperatur')

# setter navn til x-aksen
plt.xlabel('Plommetemperatur i celsius')

# setter navn til y-aksen
plt.ylabel('Koketid i minutter')

# rutenett (valgfri)
plt.grid()

# plotter grafene
plt.plot(x_verdier,y_verdier_0)
plt.plot(x_verdier,y_verdier_10)
plt.plot(x_verdier,y_verdier_100)
plt.plot(x_verdier,y_verdier_1000)
plt.plot(x_verdier,y_verdier_10000)

# viser høyden for hver linje
plt.legend(['h=0', 'h=10', 'h=100', 'h=1000', 'h=10000'])
```

Program A.7: Programmet plotter koketiden for egg avhengig av plommetemperatur.

Output:



Figur A.6: Figuren viser koketid til egg med forskjellige plommetemperaturer.

Vi ser fra grafen A.6 desto høyere er høyden, desto lengre tid tar det å koke egget. Grafen viser også at kokepunktet (plommetemperaturen) ikke overstiger fra $60 - 70$ celsius når høyden er 10000 (omtrentlig høyde for Mount Everest), og dette stemmer med det resultatet vi fikk i forrige deloppgaver.

3.4 Geometri

Oppgave Lag et program som beregner den ukjente kateten når vi kjenner hypotenus og én katet i en rettvinklet trekant.

Oppgaven er hentet fra (Bueie, 2019).

Løsning Den ukjente kateten finner vi ved å bruke Pytagoras setningen:

$$c^2 = a^2 + b^2 \tag{A.4}$$

der c er hypotenus, og a og b kateter i en rettvinklet trekant. Hypotenusen og en av katetene kjenner vi fra før, den andre kateten finner vi slik:

$$b = \sqrt{c^2 - a^2} \tag{A.5}$$

Algoritme

1 Få lengden for hypotenusen fra brukeren ved å kalle *input()*-funksjonen; konvertere input-verdien til flyttall.

2 Få lengden for den kjente kateten fra brukeren ved å kalle *input()*-funksjonen; konvertere input-verdien til flyttall.

3 Bruke formelen A.5 for å finne den andre kateten.

4 Skrive ut resultatet.

```
# importerer biblioteker/moduler
import math
import numpy as np

# får størrelse for hypotenusen
c = float(input('Oppgi lengden på hypotenusen: '))

# får størrelse for kateten
a = float(input('Oppgi lengden på den kjente kateten: '))

# bruker Pytagoras setningen for å finne den siste kateten
b = (c**2 - a**2)**0.5

'''
alternativt kan vi bruke sqrt()-funksjonen enten fra math-modulen eller NumPy-biblioteket
syntaks:
b = np.sqrt(c**2 - a**2)
b = math.sqrt(c**2 - a**2)
'''

print(f'Den siste kateten er {b:.3f} lang.')
```

Program A.8: Programmet beregner den ukjente kateten når vi kjenner hypotenus og én katet i en rettvinklet trekant.

Eksempel output:

```
Oppgi lengden på hypotenusen: 5
Oppgi lengden på den kjente kateten: 2
Den siste kateten er 4.583 lang.
```

4 Matematikk 1T

Kompetansemål etter matematikk 1T

Mål for opplæringa er at eleven skal kunne

- formulere og løyse problem ved hjelp av algoritmisk tenking, ulike problemløysingsstrategiar, digitale verktøy og programmering
- utforske strategiar for å løyse likningar, likningssystem og ulikskapar og argumentere for tenkjemåtane sine
- bruke gjennomsnittleg og momentan vekstfart i konkrete døme og gjere greie for den deriverte

Kilde

4.1 Programmering

4.1.1 Tall og variabler

Oppgave Lag et program som lagrer to tall i variabler og skriver ut summen av de to tallene.

Oppgaven er hentet fra (ProFag, 2021).

Løsning Vi bruker symboler i matematikk for å representere tall. Det samme gjelder i programmering også, med forskjell at vi bruker symboler for flere forskjellige variabler enn tall som for eksempel strenger, lister, boolske parametere, osv.

Vi definerer to tall a og b, og deretter summerer vi dem. Vi kan også få tallene fra brukeren ved å bruke *input()*-funksjonen. Det som brukeren skriver er en streng, og vi må passe på å konvertere strengen til tall, ellers vil man få feil melding ved kompilering.

Algoritme

1 Be brukeren om å oppgi et tall a, og konvertere det til flyttall.

2 Be brukeren om å oppgi et tall b, og konvertere det til flyttall.

3 Summere a og b, og lagre resultatet i en variabel *resultat*.

4 Skrive ut resultatet.

Det går an å skrive ut resultatet direkte uten å definere en ny variabel for lagring siden programmet er enkelt.

```
'''
enkelt program for å summere to tall a og b
'''
# få tallene a og b fra brukeren
a = float(input('Oppgi et tall a: '))
b = float(input('Oppgi et tall b: '))

# definere en variabel resultat
resultat = a + b

# skrive ut resultatet med to desimaler nøyaktighet
print(f'Summen av {a} og {b} er {resultat:.2f}')
```

Program A.9: Programmet summerer to tall a og b.

Eksempel output:

```
Oppgi et tall a: 3094
Oppgi et tall b: 2109
Summen av 3094.0 og 2109.0 er 5203.00
```

4.1.2 Euklidsk divisjon Divisjons teorem (Euklidsk divisjon): Gitt to heltall a og b hvor $b \neq 0$, da finnes det to unike heltall q og r slik at:

$$a = bq + r \tag{A.6}$$

og

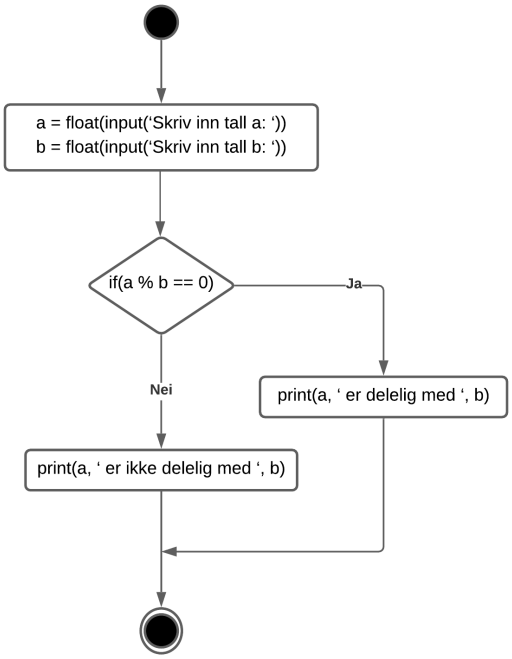
$$0 \leq r < |b|$$

Her kalles a *dividend*, b *divisor*, q *kvotient* og r *rest*.

Beregningen av kvotienten og resten fra a og b kalles divisjon eller euklidsk divisjon (I.T., udatert).

Oppgave Lag et program som tar to tall a og b, og sjekker om a er delelig med b.

Løsning Et tall er delelig med et annet tall dersom resten av divisjonen er lik null. I Python kan vi enkelt sjekke om delelighet med bruk av den aritmetiske operatoren modulus %. Et tall a er delelig med et tall b dersom $a\%b = 0$. Vi benytter dermed *if-else*-setningen for å programmere betingelsen for delelighet. Flytdiagrammet A.7 viser algoritmen for å programmere oppgaven.



Figur A.7: Flytdiagram for å sjekke om tall a er delelig med tall b

Algoritme

1 Definere en funksjon *er_delelig()*.

1.1 Bruke *input()*-funksjonen for å få tallene a og b fra brukeren, samtidig som vi konverterer tallene til flyttall.

1.2 Hvis $a\%b$ er lik null

1.2.1 skrive ut at a er delelig med b.

1.3 Ellers

1.3.1 skrive ut at a ikke er delelig med b.

2 Kalle funksjonen *er_delelig()*.

```
'''
Programmet sjekker om a er delelig med b
'''

def er_delelig():

    # ber brukeren om å skrive inn tallene a og b, konverterer tallene til float
    a = float(input('Skriv inn tall a: '))
    b = float(input('Skriv inn tall b: '))

    # sjekker om a er delelig med b
    if a % b == 0:

        # skriver ut resultatet
        print(f'Tallet {a} er delelig med {b}')

    # ellers skriver ut det motsatte
    else:
        print(f'Tallet {a} er ikke delelig med {b}')

er_delelig()
```

Program A.10: Programmet sjekker om et tall a er delelig med et annet tall b.

Eksempel output:

```
Skriv inn tall a: 4321
Skriv inn tall b: 1234
Tallet 4321.0 er ikke delelig med 1234.0
```

4.2 Algoritmisk tenkning og problemløsning

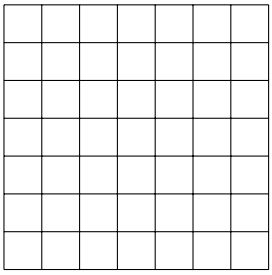
4.2.1 Kvadrattall Kvadrattall er et figurtall på formen $S_n = n^2$, der n er et heltall (Weisstein, 2002c).

De første kvadrattallene er:

$$\begin{aligned} 1^2 &= 1 \\ 2^2 &= 4 \\ 3^2 &= 9 \\ 4^2 &= 16 \\ 5^2 &= 25 \\ 6^2 &= 36 \\ &\vdots \end{aligned}$$

Vi kaller disse tallene *kvadrattall* i og med at slikt antall prikker eller kuler kan arrangeres som et geometrisk kvadrat, dvs. det n-te kvadrattallet har n prikker eller kuler i hver sin sidekant.

Oppgave Hvor mange kvadrater finnes det på figuren nedenfor?

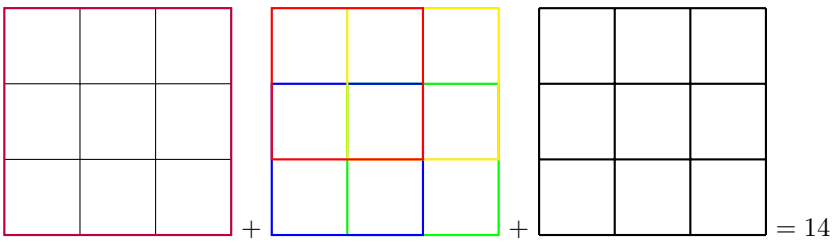


Oppgaven og problemløsnings strategien [10] er hentet fra (Kalvø mfl., 2020).

Problemløsnings strategi

- **Forstå problemet:** Denne figuren består av langt flere kvadrater enn de 49 små kvadratene. Vi kan tegne kvadrater som har 2 i høyde og bredde, vi kan tegne kvadrater som har 3 i høyde og bredde osv. Så vi må forstå at problemet er mer komplekst enn å bare telle de små kvadratene. Når vi har forstått problemet, går vi i gang med å lage en plan.
- **Lage plan:** Vi kan lage et kvadrat på 2×2, og flytte dette rundt for å telle hvor mange slike kvadrater man kan lage. Man kan også tenke at hun har 3×3, og flytte disse rundt, men dette vil kanskje ta veldig lang tid.

  Det lønner seg dermed å forenkle problemet, dvs. at vi lager et problem som er enklere å løse. Vi kan lage en figur som er delt inn i 2×2 kvadrater der har vi 4 små ruter og 1 stor rute, og til sammen $1 + 4 = 5$ kvadrater. Nå er vi på det steget som heter å gjennomføre planen.
- **Gjennomføre planen:** Vi deler inn kvadratet i 3×3 ruter, og teller hvor mange kvadrater vi kan lage.



  Vi har nå 1 stort kvadrat som er hele figuren, 4 kvadrater som er 2×2 kvadrater, og 9 små kvadrater. Vi får altså $1+4+9 = 14$ kvadrater. Det finnes en sammenheng i denne metoden; vi ser at 1, 4, og 9 er kvadrattallene. Når vi har 2×2 ruter får vi summen av de 2 første kvadrattallene, og når vi har 3×3 ruter får vi summen av de 3 første kvadrattallene. Det betyr at med 7×7 ruter, vil summen av kvadrater være lik summen av de 7 første kvadrattallene, dvs. $1 + 4 + 9 + 16 + 25 + 36 + 49 = 140$ kvadrater. Så kommer vi til det fjerde trinnet i problemløsnings strategien.
- **Se tilbake:** På dette trinnet skal vi bli sikre om at løsningen er riktig. Vi har vist at mønsteret vårt stemmer for figur med 1, 4 og 9 ruter. Vi kan se tilbake, og prøve med 4×4 ruter, og gjør vi det, ser vi at løsningen stemmer.

Vi bruker dette mønsteret for å programmere oppgaven.

[10] Problemløsnings strategien ble først definer av Pólya (1957).

Algoritme

1. Spørre brukeren om antall kvadrater for hver side.
2. Definere en variabel som kalles *resultat*, og sette den lik 0.
3. Vi kan videre bruke en *for*-løkke for å finne svaret. For hver rute $i = 1, 2, 3, ...$, så lange i ikke er lik antallet av kvadrater per side, er $resultat \mathrel{+}= i^2$.
4. Skrive ut resultatet.

Vi må være oppmerksomme på at *for*-løkken begynnes med 0, så vi må sette antall kvadrater lik $antall\_kvadrater + 1$. Alternativt kan vi skrive koden slik:

```
for i in (i+1 for i in range(antall_kvadrater)):
...
```

```
'''
Programmet finner antall kvadrater på et kvadratisk rutenett
'''

# spør brukeren om antall kvadrater per side
antall_kvadrater = int(input('Skriv inn antall kvadrat per side: '))

# definerer variabelen resultat
resultat = 0

# bruker for-løkke for å summere kvadrattallene
for i in range(antall_kvadrater + 1):
    resultat += i**2

# skriver ut resultatet
print('Antall kvadrater er ', resultat)
```

Program A.11: Programmet finner antall kvadrater på et kvadratisk rutenett.

Output:

```
Skriv inn antall kvadrat per side: 7
Antall kvadrater er  140
```

4.2.2 Palindromtall "Et palindromtall er et tall som forblir det samme når det skrives fremover eller bakover, dvs. er på formen $a_1a_2...a_2a_1$"(Weisstein, 2002b).

Oppgave Lag et program som finner hvor mange palindromtall som finnes i intervallet $[a, b]$ hvor a og b er input-verdiene gitt av brukeren.

Løsning For å løse oppgaven må vi lage en funksjon som sjekker om et tall er et palindromtall. For dette reverserer vi tallet og sjekker om tallet og det reverserte er like.

Problemløsnings strategien er hentet fra (ManBearPig, 2016).

Problemløsnings strategi Vi fjerner det siste sifferet fra tallet og legger det til revers-tallet, vi fortsetter til det opprinnelige tallet er borte, og det reverserte tallet er fullført. Metoden forutsetter at tallet er et heltall.

- **Steg 1: Isolere det siste sifferet**

  ```
  rest = tall % 10
  ```

  Restdivisjon-operatoren returnerer resten av en divisjon. I dette tilfellet deler vi tallet med 10 og returnerer resten, dvs. det siste sifferet. Vi lagrer dette tallet i et heltall-variabel kalt *rest*.

- **Steg 2: Legge rest til revers**

  ```
  revers = (revers * 10) + rest
  ```

  Vi begynner å bygge det reverserte tallet ved å legge *rest* i *revers*-variabelen. Vi multipliserer *revers* med 10, grunnen er at vi har delt tallet vårt på 10. Så for å få riktig tall, ganger vi resten med 10 hver gang vi henter den fra det opprinnelige tallet.

- **Steg 3: Fjerne det siste sifferet fra tallet**

  ```
  tall = tall // 10
  ```

  For å fjerne det siste sifferet fra tallet deler vi det med 10. Vi benytter *heltallsdivisjon*-operatoren som avrunder resultatet ned til nærmeste heltall f.eks. $244//10 = 24$

- **Gjenta steg 1-3**

  ```
  while (tall > 0)
  ```

  Gjenta denne prosessen til tallet er redusert til null og revers-tallet er fullført.

Algoritme

1 Definere en funksjon *er_palindromtall()*; funksjonen tar et heltall som parameter, og sjekker om tallet er et palindromtall eller ikke.

  1.1 Definere en variable *revers*, og sette den lik 0.

  1.2 Definere en variabel *temp*, og sette den lik tallet vi fikk som parameter. Denne variabelen bruker vi som en hjelpevariabel.

  1.3 Benytte en *while*-løkke som looper og finner palindromtallene så lenge *temp* ikke er lik null.

  1.4 Definere en variabel *rest*, og sette den lik *temp*%10.

  1.5 Multiplisere *revers* med 10, og legge *rest* til den; vi setter *revers* lik dens nye verdi.

  1.6 Bruke heltallsdivisjon, og dele *temp* på 10; vi setter *temp* lik dens nye verdi.

  1.7 Sjekke om tallet og dets reverserte er like; funksjonen returnerer *True* dersom de er like, og *False* ellers.

2 Definere *main()*-funksjonen for å finne palindromtallene på intervallet $[a, b]$.

  2.1 Få input-verdiene a og b fra brukeren; og konvertere dem til heltall.

  2.2 Benytte en *for*-løkke som looper fra a til b.

    2.2.1 Sjekke om tallet på indeks i er et palindromtall ved å kalle *er_palindromtall()*-funksjonen.

      2.2.1.1 Skrive ut tallet hvis tallet er et palindromtall.

3 Kalle *main()*-funksjonen.

```python
'''
Programmet finner palindromtall på et intervall [a, b]
'''
# definerer funksjon som sjekker om et tall er palindrom
def er_palindromtall(tall: int):

    # definerer en variabel for den reversen av tallet
    revers = 0

    # temp brukes som hjelpetall
    temp = tall

    # looper så lenge tallet er større enn 0
    while temp > 0:

        # deler tallet på 10, og finner resten
        rest = temp % 10

        # gang revers med 10 og legg rest til den
        revers = revers * 10 + rest

        # sett tallet lik tallet del på 10
        temp //= 10

    # hvis tallet er lik revers returnerer true, ellers false
    return (tall == revers)


# main-funksjonen skriver ut alle palindromtallene i intervallet [a, b]
def main():

    # får startverdi fra brukeren
    a = int(input('Oppgi startverdien: '))

    # får sluttverdi fra brukeren
    b = int(input('Oppgi sluttverdien: '))

    # looper fra a til b; legger 1 til b siden løkken begynnes fra indeks 0
    for i in range(a, b+1):

        # sjekker om tallet på indeks i er et palindromtall
        if er_palindromtall(i):

            # skriver ut tallet, end = ' ' setter et mellomrom etter tallet (neste tall kommer etter)
            print(i, end = ' ')

main()
```

Program A.12: Programmet finner palindromtall på et intervall [a, b].

Eksempel output:

```
Oppgi startverdien: 100
Oppgi sluttverdien: 1000
101 111 121 131 141 151 161 171 181 191 202 212 222 232 242 252 262 272 282 292 303 313 323 333
↪  343 353 363 373 383 393 404 414 424 434 444 454 464 474 484 494 505 515 525 535 545 555 565
↪  575 585 595 606 616 626 636 646 656 666 676 686 696 707 717 727 737 747 757 767 777 787 797
↪  808 818 828 838 848 858 868 878 888 898 909 919 929 939 949 959 969 979 989 999
```

4.3 Andregradslikninger

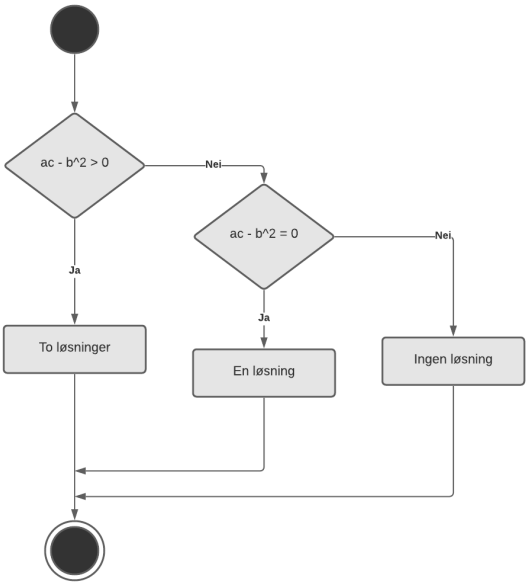
4.3.1 abc-formelen Andregradslikninger kan generelt skrives på formen:

$$ax^2 + bx + c = 0 \tag{A.7}$$

og kan løses med bruk av abc-formelen:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{A.8}$$

Denne typen funksjoner har enten to løsninger, en løsning eller ingen løsning. Problemet kan derfor programmeres ved hjelp av betingelser. Bueie (2019, s. 65) bruker et flytskjema A.8, og viser klart hvordan man kan skrive et program som løser andregradslikninger ved hjelp av *abc*-formelen.



Figur A.8: Antall mulige løsninger for andregradsligninger

Source: Bueie, 2019

Oppgave Lag et program som løser en andregradslikning på formen A.7 gitt koeffisientene a, b og c.

Løsningsforslaget er hentet fra (Bueie, 2019, s. 66).

Løsning For å løse oppgaven bruker vi *if-elif-else*-setning siden vi har tre mulige løsninger.

Algoritme

1 Definere en funksjon *løs_andregrads_likning()*.

1.1 Få koeffisientene a, b og c fra brukeren som input; konvertere input-verdiene til flyttall.

1.2 Hvis $b^2 - 4ac > 0$, har likningen to løsninger der x er gitt ved formelen A.8.

1.2.1 Definere en variabel x_1, og sette den lik $\frac{-b+\sqrt{b^2-4ac}}{2a}$.

1.2.2 Definere en variabel x_2, og sette den lik $\frac{-b-\sqrt{b^2-4ac}}{2a}$.

1.2.3 Skrive ut svarene med bruk av *print()*-funksjonen.

1.3 Eller hvis $b^2 - 4ac = 0$, har likningen én løsning.

1.3.1 Definere en variabel x_1 gitt ved $\frac{-b}{2a}$.

1.3.2 Skrive ut resultatet.

1.4 Ellers har likningen ingen løsning;

1.4.1 Skrive ut at likningen ikke har noen løsninger.

2 Kalle *løs_andregrads_likning()*-funksjonen.

```
'''
Programmet løser andregradslikningen f(x) vha. abc-formelen
'''

def løs_andregrads_likning():

     # får koeffisientene a, b og c fra brukeren
    a = float(input('Skriv inn a-koeffisienten:'))
    b = float(input('Skriv inn b-koeffisienten:'))
    c = float(input('Skriv inn c-koeffisienten:'))

    # hvis b^2-4ac > 0, har likningen to løsninger
    if((b**2)-(4*a*c)) > 0:
        x_1 = (-b + (b**2-4*a*c)**0.5)/(2*a)
        x_2 = (-b - (b**2-4*a*c)**0.5)/(2*a)

        print('Løsningene på andregradslikningen er x_1 = ', x_1, ' og x_2 = ', x_2)

    # hvis b^2-4ac = 0, har likningen én løsning
    elif((b**2)-(4*a*c)) == 0:
        x_1 = (-b/2*a)

        print('Løsningen på andregradslikningen er x = ', x_1)

    # ellers har ikke likningen noen løsning
    else:
        print('andregradslikningen har ingen reelle løsninger')

løs_andregrads_likning()
```

Program A.13: Programmet løser andregradslikningen $f(x)$ vha. abc-formelen.

Eksempel output:

```
Skriv inn a-koeffisienten:1
Skriv inn b-koeffisienten:2
Skriv inn c-koeffisienten:-3
Løsningene på andregradslikningen er x_1 =  1.0  og x_2 =  -3.0
```

4.3.2 Halverings metoden Når vi skal løse en andregradslikning med halverings metoden, ordner vi likningen med 0 på høyre side:

$$f(x) = 0$$

I halverings metoden tar vi utgangspunkt i et intervall $[a, b]$, og finner et midtpunkt m ved å halvere intervallet. Se A.42.

Etter halveringen, sjekker vi om $f(a)$ eller $f(b)$ ligger på motsatt side av x-aksen som $f(m)$ (dvs. om de har forskjellige fortegn). Hvis $f(b)$ og $f(m)$ ligger på samme side, setter vi $b = m$, og halverer intervallet på nytt. Hvis dette ikke er tilfelle så vet vi at $f(a)$ og $f(m)$ ligger på samme side av x-aksen, og setter $a = m$. Vi fortsetter med å halvere så langt vi kommer til et intervall

som har en lengde som er mindre enn en gitt nøyaktighetsverdi. Midtpunktet til dette intervallet, vil være vår løsning.

Oppgave Skriv et program i Python som løser likningen $x^2 - 3 = 0$ for $x \in [1, 2]$. Bruk halverings metoden med en tusendels nøyaktighet.

Oppgaven og løsningsforslaget er hentet fra (Kalvø mfl., 2020).

Algoritme

1 Definere en funksjon $f()$ som returnerer likningen $f(x) = x^2 - 3$.

2 Definere nedregrense $a = 1$, øvregrense $b = 2$, og nøyaktighets verdi $toleranse = 0.001$ som konstanter.

3 Definere midtpunktet m som vi finner ved bruk av formelen A.42.

4 Bruke en *while*-løkke, og fortsette halveringen så lenge $b - a > 0.001$.

4.1 Sjekke om $f(a)$ ligger på motsatt side av x-aksen som $f(m)$, for dette sjekker vi om $f(a) * f(m) < 0$, grunnen er at dersom de to verdiene ligger på samme side vil produktet av dem være positivt, ellers er produktet negativt.

4.1.1 hvis ja, da er $b = m$.

4.2 Ellers

4.2.1 er $a = m$.

4.3 Finne midtpunktet m for det nye intervallet ved å bruke formelen A.42 på nytt.

5 Skrive ut resultatet.

```
'''
Programmet løser andregradslikningen f(x) vha. halverings metoden
'''

def f(x):
    return x**2-3

# definerer nedregrense a, øvregrense b og nøyaktighetsverdien
a = 1
b = 2
toleranse = 0.001

# definerer midtpunktet m
m = (a+b)/2

# bruker while-løkke for halvering, fortsetter så langt b-a > toleranse
while b-a > toleranse:

# dersom produktet av f(a) og f(m) er mindre enn 0 betyr det at de ligger på motsatt side av x-aksen
    if f(a) * f(m) < 0:
        # setter b lik m
        b = m

        # dersom f(m) og f(b) ligger på motsatt side av x-aksen
    else:
        # setter a lik m
        a = m

    # og finner midtpunktet på nytt
    m = (a+b)/2


# Skriver ut svaret med tre siffer nøyaktighet
print(f'Løsningen av likningen er x = {m:.3f}')
```

Program A.14: Programmet løser andregradslikningen $f(x)$ vha. halverings metoden.

Output:

```
Løsningen av likningen er x = 1.732
```

4.4 Vekstfart og derivasjon

4.4.1 Gjennomsnittlig vekstfart

Oppgave Vi fyller kakao i en kopp. Kakaotemperaturen etter x minutter er gitt ved:

$$T(x) = 65 * 0.97^x + 20 \quad x \in [0, 70]$$

a Tegn grafen til $T(x)$.

b Hva er kakaotemperaturen når vi fyller kakao i koppen?

c Hvor mye faller temperaturen fra 30 til 60 minutter etter at vi fylte koppen?

d Hva er den gjennomsnittlige vekstfarten i intervallet $[30, 60]$?

Oppgaven er hentet fra (Kalvø mfl., 2020).

Løsning Vi definerer først en funksjon $f(x)$ som returnerer likningen. Denne funksjonen skal vi benytte i andre del-oppgaver også.

```
def f(x):
    return 65*0.97**x+20
```

Program A.15: Kodesnutt som definerer funksjonen $f(x)$.

a) **Algoritme**

1 Importere biblioteket *Numpy* og modulen *matplotlib.pyplot*.

2 Definere en funksjon *tegn_graf()*.

2.1 Kalle *linspace()*-funksjonen fra *NumPy*-biblioteket for å definere x-verdier, (fra, til, antall sampler).

2.2 Sette $y = f(x)$ hvor x-verdier er parameter til funksjonen f.

2.3 Kalle *plot()*-funksjonen fra *matplotlib.pyplot* for å tegne grafen.

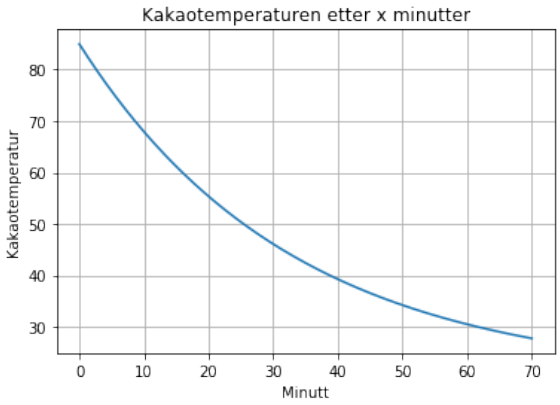
Man kan gjerne sette x-label, y-label og tittel for plottet, dette hjelper med å forstå grafen bedre. Det er også mulig å sette rutenett for plottet.
For disse kaller vi henholdsvis *xlabel('x-label')*, *ylabel('y-label')*, *title('tittel')*, og *grid()* fra *matplotlib.pyplot*-modulen.

3 Kalle *tegn_graf()*-funksjonen.

```
# a: tegner grafen til f(x)

# importerer biblioteker
import matplotlib.pyplot as plt
import numpy as np

def tegn_graf():
    # definerer x-verdier
    x = np.linspace(0, 70, 1000)

    # alternativ kan vi bruke numpy.arange([start, ]stop, [step, ]dtype=None, *, like=None)
    # x = np.arange(0, 70, .001)

    # definerer y-verdier
    y = f(x)

    # tegner grafen
    plt.xlabel('Minutt')
    plt.ylabel('Kakaotemperatur')
    plt.title('Kakaotemperaturen etter x minutter')
    plt.grid()
    plt.plot(x, y)

tegn_graf()
```

Program A.16: Programmet plotter grafen til funksjonen $f(x)$.

Output:



Figur A.9: Grafen viser kakaotemperaturen etter x minutter.

På grunn av at de neste deloppgavene er relatert til hverandre, løser vi resten av oppgaven ved å lage en funksjon for hver deloppgave, og kalle funksjoner ved behov i andre deloppgavene. Dermed legges det inn funksjonen som løsning uten utskrift, og vi skriver ut resultatet til slutt.

b) Når vi fyller koppen er $x = 0$, verdien ved denne tiden kan vi finne ved å skrive ut $f(0)$. En mer fleksibel måte er å lage en funksjon som mottar x-verdi, og returnerer resultatet.

Algoritme

1 Definere en funksjon *finn_temp()*.
  1.1 Få x-verdien som input fra brukeren, samtidig konverterer vi input-verdien til flyttall.
  1.2 Finne temperaturen ved å kalle *f()*-funksjonen hvor x er parameter til funksjonen.
  1.3 Returnere temperaturen samt x-verdien.

```
# b: definerer en funksjon som finner temperatur ved et gitt tidspunkt (x-verdi)

def finn_temp():

    # få x-verdi fra brukeren via input
    x = float(input('Skriv inn en x-verdi: '))

    # bruker funksjonen vi har definert til å finne temperaturen ved tiden x
    temp = f(x)

    # returnerer både x-verdi og temperaturen
    return x, temp
```

Program A.17: Programmet finner temperatur ved et gitt tidspunkt.

c) Vi lager en funksjon som bruker *finn_temp()*-funksjonen ved hvert tidspunkt, finner differansen mellom dem, og returnerer resultatet. Vi returnerer både tidsdifferansen og temperaturdifferansen, for vi har behov for disse to for å løse deloppgaven d.

Algoritme

1 Definere en funksjon *finn_differanse()*.
  1.1 Kalle *finn_temp()*-funksjonen to ganger; funksjonen *finn_temp()* returnerer temperaturen gitt ved en x-verdi, og selve x-verdien.

1.2 Finne differansen mellom x-verdiene (tidsendring).
1.3 Finne differansen mellom temperatur-verdiene (temperaturendring).
1.4 Returnere tidsendring og temperaturendring.

```python
# c: funksjon for å finne differansen av temperatur mellom to tidspunkter

def finn_differanse():

    # vi kaller funksjonen finn_temp for å finne først temperatur-verdiene
    x_1, temp_1 = finn_temp()
    x_2, temp_2 = finn_temp()

    # vi beregner differansen mellom x-verdiene også dette er pga vi vil bruke funksjonen mest effektivt
    x_differanse = x_2 - x_1

    # finner differansen av temperatur-verdiene
    temp_differanse = temp_2 - temp_1

    return x_differanse, temp_differanse
```

Program A.18: Programmet finner temperaturendring mellom to tidspunkter.

d) Gjennomsnittlig vekstfart på et tidsintervall er lik temperatur per tidspunkt. Dette finner vi ved følgende formel:

$$gjennomsnittlig\, vekstfart = \frac{temperaturendring}{tidsendring} = \frac{\Delta y}{\Delta x} \tag{A.9}$$

Vi kaller *finn_differanse()*-funksjonen for å finne de verdiene vi har behov for, og returnerer resultatet.

Algoritme

1 Definere en funksjon *finn_vekstfart()*.
1.1 Finne temperatur- og tids differansen ved å kalle *finn_differanse()*-funksjonen.
1.2 Finne gjennomsnittlig vekstfart ved å bruke formelen A.9.
1.3 Returnere gjennomsnittlig vekstfart.

```python
# d: definerer en funksjon som finner vekstfart på et intervall, vekstfart=temperatur per minutt

def finn_vekstfart():

    # kaller funksjonen finn_differanse for å finne differanse-verdier
    x_differanse, temp_differanse = finn_differanse()

    # finner gjennomsnittlig vekstfart
    snitt_vekstfart = temp_differanse/x_differanse
    return snitt_vekstfart
```

Program A.19: Programmet finner vekstfart på et tidsintervall.

Klientprogram:

```python
# klientprogram for å løse oppgavesettet

def main():

    # b. Hva er kakaotemperaturen når vi fyller kakao i koppen?
    print('Oppgave b: ')

    # kaller finn_temp()-funksjonen for å finne temperaturen ved tiden x=0
    x, temp = finn_temp()

    # skriver ut resultatet med tre-siffer-nøyaktighet
    print(f'Kakaotemperaturen ved tiden x = {x} er: {temp:.3f}')

    # c. Hvor mye faller temperaturen fra 30 til 60 minutter etter at vi fylte koppen?
    print('\nOppgave c: ')

    # finner temperaturdifferansen med å kalle finn_differanse()-funksjonen,
    x_differanse, temp_differanse = finn_differanse()

    # bruker abs (absoluttverdi) for å skrive ut tallet positivt, (med tre-siffer-nøyaktighet)
    print(f'Etter {x_differanse} minutter, faller temperaturen {abs(temp_differanse):.3f} grader.')

    # d. Hva er den gjennomsnittlige vekstfarten i intervallet [30, 60]?
    print('\nOppgave d: ')

    # finner gjennomsnittlig vekstfart ved å kalle finn_vekstfart()-funksjonen
    snitt_vekstfart = finn_vekstfart()

    # skriver ut resultatet med tre-siffer-nøyaktighet
    print(f'Gjennomsnittlig vekstfarten er {snitt_vekstfart:.3f}')

main()
```

Program A.20: Klientprogram for å løse oppgaven.

Output:

```
Oppgave b:
Skriv inn en x-verdi: 0
Kakaotemperaturen ved tiden x = 0.0 er: 85.000

Oppgave c:
Skriv inn en x-verdi: 30
Skriv inn en x-verdi: 60
Etter 30.0 minutter, faller temperaturen 15.613 grader.

Oppgave d:
Skriv inn en x-verdi: 30
Skriv inn en x-verdi: 60
Gjennomsnittlig vekstfarten er -0.520
```

4.4.2 Numerisk derivasjon

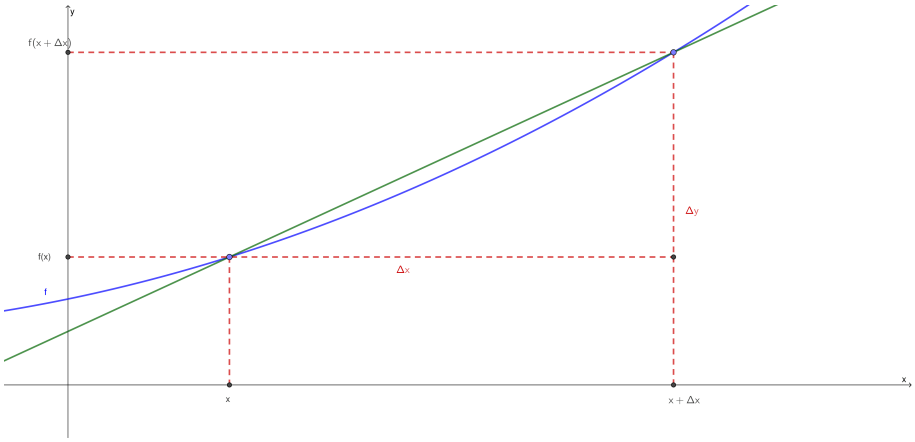
Oppgave La f være funksjonen gitt ved:

$$x^2 + 3x + 6$$

Regn ut en tilnærmingsverdi for $f'(5)$ med $\Delta x = 0.01$.

Oppgaven og løsningsforslaget er hentet fra (Kalvø mfl., 2020).

Løsning Når vi vil finne den deriverte av en funksjon (gjennomsnittlig vekstfart), deler vi endring i y på endring i x over et intervall. Se figur A.10.



Figur A.10: Illustrasjon for gjennomsnittlig vekstfart for funksjonen f.

$$gjennomsnittlig\ vekstfart = \frac{\Delta y}{\Delta x} = \frac{f(x + \Delta x) - f(x)}{\Delta x} \tag{A.10}$$

Algoritme

1. Definere en funksjon $f()$ som returnerer likningen $f(x)$; funksjonen tar x som parameter.
2. Definere konstanter $\Delta x = 0.01$, og $x = 5$.
3. Finne $f'(5)$ ved bruk av formelen A.10.
4. Skrive ut resultatet.

```
'''
Programmet finner den deriverte til f(x) numerisk
'''

def f(x):
    return x**2+3*x+6

# definerer konstanter
delta_x = 0.01
x = 5

# finner den deriverte av f i punkt x=5
derivert = (f(x+delta_x) - f(x)) / delta_x

# skriver ut resultatet med 3 siffer nøyaktighet
print(f'Tilnærmings verdien er lik: {derivert:.3f}')
```

Program A.21: Programmet finner numerisk den deriverte til $f(x)$.

Output:

```
Tilnærmings verdien er lik: 13.010
```

5 Matematikk 2P

Kompetansemål etter matematikk 2P

Mål for opplæringa er at eleven skal kunne

- forklare og bruke prosent, prosentpoeng og vekstfaktor til modellering av praktiske situasjonar med digitale verktøy
- utforske strategiar for å løyse likningar, likningssystem og ulikskapar og argumentere for tenkjemåtane sine
- vurdere val knytte til personleg økonomi og reflektere over konsekvensar av å ta opp lån og å bruke kredittkort

Kilde

5.1 Likninger

"To likninger med samme ukjente størrelser, kalles for et likningssett"(Kristensen & Aanensen, 2020). Vi løser et likningssett ved å finne verdier som passer for alle likningene i likningssettet.

Oppgave Løs følgende likningssettet:

$$\begin{cases} f_1(x) = sin(x) \\ f_2(x) = cos(x) \end{cases} \tag{A.11}$$

Løsning For å løse oppgaven setter vi de to likningene lik hverandre:

$$sin(x) = cos(x) \iff sin(x) - cos(x) = 0$$

Vi kaller denne funksjonen $f()$, og løser den ved hjelp av *fsolve()*-funksjonen fra *scipy.optimize*-modulen. Funksjonen tar funksjonen $f()$ og en startgjett x_0 som parametere og returnerer x-verdien til roten dvs. skjæringspunktet mellom de to funksjonene. Vi setter denne verdien inn i en av funksjonene for å finne y-verdien til skjæringspunktet. x_0 kan være enten ett tall eller en n-dimensjonal *array* av flere tall (The-SciPy-community, 2021b).

Etter at vi løste oppgaven algebraisk, plotter vi grafen til de to funksjonene, samt skjæringspunktene mellom dem.

Algoritme

1 Importere biblioteket *Numpy*, modulen *pyplot*, og funksjonen *fsolve()* fra *scipy.optimize()*-modulen.

2 Definerer en funksjon *f_1()* som returnerer $sin(x)$; funksjonen tar x som parameter.

3 Definerer en funksjon *f_2()* som returnerer $cos(x)$; funksjonen tar x som parameter.

4 Definere en funksjon $f()$ som returnerer $sin(x) - cos(x)$; funksjonen tar x som parameter.

5 Definere startgjett x_0, vi velger her to startgjetter $[0, 4]$.

6 Finne x-koordinat til skjæringspunkter ved hjelp av *fsolve()*-funksjonen, vi definerer en variabel x_1 for lagring av resultatet.

7 Sette x_1 i *f_1()*-funksjonen, og finne y_1.

8 Bruke en *for*-løkke som looper to ganger; antall ganger løkken looper setter vi lik lengden til startverdi-listen.

  8.1 Skrive ut koordinaten til skjæringspunktet (x_i, y_i).

9 Lage en liste av x-verdier ved å kalle *linspace()*-funksjonen.

10 Sette x-verdiene i funksjonene *f_1()* og *f_2()*, og lage to sett av y-verdier.

11 Sette tittel for plottet ved å kalle *title()*-funksjonen fra *matplotlib.pyplot*.

12 Sette label for x- og y-aksen ved å kalle henholdsvis *xlabel()*- og *ylabel()*-funksjonen fra *matplotlib.pyplot*.

13 Plotte funksjonene og skjæringspunktene ved å kalle *plot()*-funksjonen; funksjonen tar x- og y-verdier som parametere. Det er også mulig å legge til farge for grafen, syntaksen kan se slik ut:

```
plot(x-verdi, y-verdi, 'g')
```

her 'g' står for grønn-farge.

14 Sette rutenett for plottet ved å kalle *grid()*-funksjonen.

15 Sette label for grafene ved å kalle *legend()*-funksjonen; funksjonen tar en liste av strenger (label) som parameter.

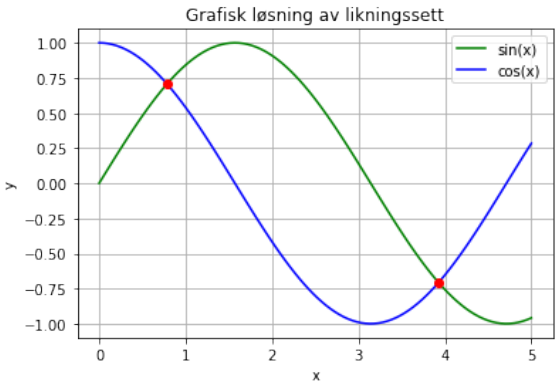
16 Vise plottet ved å kalle *show()*-funksjonen.

```python
'''
programmet viser hvordan løse et likningssett algebraisk og grafisk
'''

# importerer biblioteker
from scipy.optimize import fsolve
import matplotlib.pyplot as plt
import numpy as np

# definerer en funksjon f_1 som returnerer sin(x)
def f_1(x):
    return np.sin(x)

# definerer en funksjon f_2 som returnerer cos(x)
def f_2(x):
    return np.cos(x)

'''
funksjon f er definert som f_1 = f_ 2 -> sin(x) = cos(x)
parameter: x
returnerer f_1 - f_2
'''
def f(x):
    return f_1(x) - f_2(x)

# start gjett av rot
x_0 = [0, 4]

# finner x-verdi til skjæringspunkt mellom f_1 og f_2
x_1 = fsolve(f, x_0)

# setter x_1 i f_1 og finner y_1
y_1 = f_1(x_1)

# skriver ut koordinater til skjæringspunkter
for i in range(len(x_0)):
    print(f'({x_1[i]}, {y_1[i]})')


# data for plotting

# x_verdier for plottet
x = np.linspace(0, 5, 100)

# y_verdier for plottet
y_verdier1 = f_1(x)
y_verdier2 = f_2(x)

# setter tittel for plottet
plt.title('Grafisk løsning av likningssett')

# setter x- og y-label for aksene
plt.xlabel('x')
plt.ylabel('y')

# plotter funksjoner sin(x) og cos(x) samt skjæringspunkter
plt.plot(x, y_verdier1, 'g', x, y_verdier2, 'b', x_1, y_1,'ro')

# setter rutenett for plottet
plt.grid()

# setter label for grafer
plt.legend(['sin(x)', 'cos(x)'])

# viser grafen
plt.show()
```

Program A.22: Programmet løser et likningssett algebraisk og grafisk.

Output:

```
(0.7853981633882219, 0.7071067811800235)
(3.9269908169864465, -0.7071067811859854)
```



Figur A.11: Figuren viser grafisk løsning av likningssettet A.11 på intervallet $[0, 5]$.

Vi ser både fra outputen og grafen A.11 at det finnes to skjæringspunkter mellom 0 og 5, dvs. likningssettet har to løsninger $x = 0.79$ og $x = 3.93$ på dette intervallet.

5.2 Prosent og vekstfaktor

5.2.1 Prosentregning

Oppgave En butikk gir kunden 30% rabatt på totalprisen dersom kunden kjøper tre eller flere enheter av et spesielt produkt. Lag et program som beregner hvor mye kunden skal betale avhengig av hvor mange enheter kunden kjøper.

Oppgaven er hentet fra (Bueie, 2019).

Løsning Kunden får rabatt for kjøp av et produkt dersom antall produkter er mer enn 3. Dette er en betingelse som vi programmerer den ved hjelp av *if-else*-setning.

Algoritme

1 Definere en funksjon *beregn_beløp()*, funksjonen tar pris og antall produkter som parametere.

- 1.1 Definere konstanten *rabattfaktor*.
- 1.2 Definere en variabel *beløp* for å lagre det endelige beløpet i.
- 1.3 Hvis kunden kjøper mer enn 3 produkter
  - 1.3.1 beregne rabatt for kjøpet,
  - 1.3.2 trekke rabatten fra totalprisen, og lagre resultatet i *beløp*-variabelen.
- 1.4 Ellers
  - 1.4.1 beregnes beløpet med normalpris.

1.5 Returnere beløpet.

2 Definere *main()*-funksjonen som klientprogram.

2.1 Få prisen for produktet fra brukeren, og konvertere den til flyttall.
2.2 Få antall kjøpte produkter fra brukeren, og konvertere den til heltall.
2.3 Beregne beløpet ved å kalle *beregn_beløp()*-funksjonen.
2.4 Skrive ut resultatet.

3 Kalle *main()*-funksjonen.

```
'''
Programmet beregner sluttbeløp for kjøp av et produkt ved gitt rabattfaktor
'''

def beregn_beløp(pris, antall):

    # definerer rabattfaktoren som konstant
    rabattfaktor = 0.3

    # definerer en variabel totalpris for å lagre prisen
    beløp = 0

    # dersom kunden kjøper mer enn 3 av et produkt
    if antall > 3:

        # rabatt som skal trekkes ut fra totalprisen
        rabatt = antall * pris * rabattfaktor

        # beløp etter rabattberegning
        beløp = (antall * pris) - rabatt

    # ellers betaler kunden normal pris
    else:

        # beløp uten rabatt
        beløp = antall * pris

    # returnerer beløpet
    return beløp

def main():

    # får prisen for produktet fra brukeren
    pris = float(input('Oppgi pris for produktet: '))

    # får antall av et produkt
    antall = int(input('Oppgi antall produkter: '))

    # beregner beløpet ved å kalle beregn_beløp()-funksjonen
    beløp = beregn_beløp(pris, antall)

    # skriver ut beløpet
    print(f'Totalpris for ditt kjøp er {beløp:.2f} kr.')

main()
```

Program A.23: Programmet beregner sluttbeløp for kjøp av et produkt ved gitt rabattfaktor.

Eksempel output:

```
Oppgi pris for produktet: 23.5
Oppgi antall produkter: 5
Totalpris for ditt kjøp er 82.25 kr.
```

5.2.2 Vekstfaktor

Oppgave Verdien til en leilighet er 2 500 000 kr., og den øker med 12% hvert år. Hva var verdien til leiligheten for 4 år siden?

Oppgaven er hentet fra (Kalvø mfl., 2020).

Løsning Det er fornuftig å alltid skrive kode så fleksibel som mulig, på den måten kan vi bruke én og samme kode for å løse flere andre oppgaver av samme type. Så vi skal i denne oppgaven tilpasse løsningen slik at den kan benyttes på flere oppgaver av denne typen.

Beregning av vekstfaktoren

- Hvis verdien minker, er vekstfaktoren gitt ved: $a = 1 - a$
- Hvis verdien øker, er vekstfaktoren gitt ved: $a = 1 + a$

Verdien etter en periode t, beregnes slik:

- Verdien etter $1 * t = startverdi * a$
- Verdien etter $2 * t = startverdi * a * a$
- ...
- Verdien etter $n * t = startverdi * a^n$

Dersom vi vil finne verdien for n perioder siden, beregner vi verdien slik:

$$resultat = \frac{startverdi}{a^n}$$

Følgende informasjon skal tas via input:

- Startverdi
- Vekstfaktor
- Antall perioder

Vi trenger å vite om verdien *avtar* eller *øker*, og om vi vil beregne verdien i *fremtiden* eller *fortiden*. Når det gjelder programmering av slike tilfeller, trenger vi å spørre brukeren om det, og avhengig av input-svaret, regner vi ut verdien.

Algoritme

1. Få startverdi fra brukeren via *input()*-funksjonen, og konvertere verdien til flyttall.
2. Få vekstfaktoren i prosent fra brukeren via *input()*-funksjonen, og konvertere verdien til flyttall.

3. Spørre brukeren om verdien avtar

   3.1 Hvis ja er vekstfaktoren lik $1 - a$.

4. Ellers

   4.1 er vekstfaktoren lik $1 + a$.

5. Få antall perioder fra brukeren, og konvertere verdien til heltall.

6. Spørre brukeren om hun/han vil finne verdien i fortiden.

   6.1 Hvis ja, er resultatet lik $startverdi/a^n$.

7. Ellers

   7.1 er resultatet lik $startverdi * a^n$.

8. Skrive ut resultatet.

```
'''
Programmet beregner prisen til en leilighet ved gitt pris, periode, og rentefot
i fortiden og fremtiden
'''

# får startverdien fra brukeren (samtidig konverterer streng til float)
startverdi = float(input('Skriv inn startverdi: '))

# får vekstfaktoren fra brukeren, og deler den på 100 siden vekstfaktoren er i prosent
a = float(input('Skriv inn vekstfaktoren i prosent: '))/100

# spør brukeren om verdien avtar
q_1 = input('Avtar verdien? (ja/nei)')

# sjekker både med stor og små bokstav for å unngå feil
if q_1 == 'Ja' or q_1 == 'ja':
    a = 1 - a                   # vekstfaktor dersom verdien avtar
else:
    a += 1                      # vekstfaktor dersom verdien øker

# spør brukeren om antall perioder
n = int(input('Skriv inn antall perioder t: '))

# spør brukeren om han vil beregne verdien i fortiden
q_2 = input('Vil du finne verdien i fortiden: (ja/nei)')

if q_2 == 'Ja' or q_2 == 'ja':
    resultat = startverdi/a**n   # sluttverdi i fortiden
else:
    resultat = startverdi*a**n   # sluttverdi i fremtiden

# skrive ut resultat
print(f'Verdien til leiligheten var/er {resultat:.3f} kroner.')
```

Program A.24: Programmet beregner prisen til en leilighet ved gitt pris, periode, og rentefot i fortiden og fremtiden.

Output:

```
Skriv inn startverdi: 2500000
Skriv inn vekstfaktoren i prosent: 12
Avtar verdien? (ja/nei)nei
Skriv inn antall perioder t: 4
Vil du finne verdien i fortiden: (ja/nei)ja
Verdien til leiligheten var/er 1588795.196 kroner.
```

5.3 Økonomi

5.3.1 Forrentning

Oppgave Lag et program som beregner differansen mellom årlig og kontinuerlig forrentning, og som runder av differansen til to desimaler.

Oppgaven er hentet fra (Bueie, 2019).

Løsning Formel for årlig forrentning er gitt ved:

$$k_n = k_0 * (1 + r)^n \tag{A.12}$$

hvor k_n er beløpet etter n år, k_0 er innskutt beløp, og r er rentefoten.

Formel for kontinuerlig forrentning er gitt ved:

$$k_n = k_0 * e^{rn} \tag{A.13}$$

For å løse oppgaven, beregner vi årlig og kontinuerlig forrentning med gitte verdier fra brukeren, og så finner vi differansen mellom de to forrentningene.

Algoritme

1 Importere *math*-modulen.

2 Definere en funksjon *finn_differanse_forrentning()* som tar innskutt beløp, rentefot, og antall år som parametere.

- 2.1 Omforme rentefoten fra prosent til desimaltall.
- 2.2 Beregne årlig og kontinuerlig forrentning ved å bruke henholdsvis formlene A.12 og A.13.
- 2.3 Finne differansen mellom de to forrentningene; Vi tar absoluttverdi av differansen for å unngå negative verdier ved å kalle *abs()*-funksjonen; runder av resultatet til to desimaler ved å kalle *round()*-funksjonen.
- 2.4 Returnere årlig forrentning, kontinuerlig forrentning, og differansen.

3 Definere *main()*-funksjon som klientprogram.

- 3.1 Be brukeren om å oppgi verdiene innskutt beløp, rentefot og antall år; konvertere verdiene til flyttall.
- 3.2 Beregne årlig og kontinuerlig forrentning og differansen mellom dem ved å kalle *finn_differanse_forrentning()*-funksjonen.
- 3.3 Skrive ut resultatet.

4 Kalle *main()*-funksjonen.

```python
'''
Programmet beregner differansen mellom årlig og kontinuerlig forrentning
'''

# importerer biblioteket math
import math

'''
funksjon for å finne differansen mellom årlig og kontinuerlig forrentning
funksjonen tar innskudd:k_0, rentefot:r, og antall år:n som parametere
funksjonen returnerer differansen med to desimaler nøyaktighet
'''
def finn_differanse_forrentning(k_0, r, n):

    # omformer fra prosent til desimal
    r = r/100

    # beregner årlig og kontinuerlig forrentning
    aarlig_forrentning = k_0 * (1 + r)**n
    kontinuerlig_forrentning = k_0 * math.e**(r*n)

    # finner differansen og runder av til to desimaltall; vi tar absoluttverdi for å unngå negative tall
    differanse = round(abs(aarlig_forrentning - kontinuerlig_forrentning), 2)

    # returnerer de beregnede verdier
    return aarlig_forrentning, kontinuerlig_forrentning, differanse

def main():

    # får verdier som trengs fra brukeren
    k_0 = float(input('Oppgi innskutt beløp: '))
    r = float(input('Oppgi rentefoten: '))
    n = float(input('Oppgi antall år: '))

    # beregner årlig og kontinuerlig forrentning og differansen mellom dem
    aarlig_forrentning, kontinuerlig_forrentning, differanse = finn_differanse_forrentning(k_0, r, n)

    # skriver ut resultatet
    print(f'Årlig forrentning: {round(aarlig_forrentning)} kr.\
    \nKontinuerlig forrentning: {round(kontinuerlig_forrentning)} kr.\
    \nDifferansen mellom årlig og kontinuerlig forrentning: {differanse}')

main()
```

Program A.25: Programmet beregner differansen mellom årlig og kontinuerlig forrentning.

Eksempel output:

```
Oppgi innskutt beløp: 5000
Oppgi rentefoten: 4
Oppgi antall år: 3
Årlig forrentning: 5624 kr.
Kontinuerlig forrentning: 5637 kr.
Differansen mellom årlig og kontinuerlig forrentning: 13.16
```

5.3.2 Sparing

Oppgave Kari har 100 000 kroner på sparekontoen, som hun har tjent på sommerjobber de siste årene. Hun planlegger å spare disse pengene i banken for å få råd til leilighet en dag. For å få råd til leilighet trenger hun 300 000 kroner i egenkapital.

a Beregn hvor mange år tar det før Kari har nok penger med en rente på 3%.

b Kari bestemmer seg for å spare 15 000 kroner til per år. Modifiser programmet, hvor mange år tar det nå?

Oppgaven er hentet fra (ProFag, 2021).

Løsning

a Når man setter penger på en sparekonto i en bank, får hun en *avkastning* på innskuddet i form av renter. Rentefoten forteller hvor stor del av innskuddet man får som rente for en viss periode.

Kari setter et beløp på 100 000 kroner i banken til en fast rente på 3% per år. Vekstfaktoren er:

$$1 + 0.03 = 1.03$$

Etter ett år vil Kari ha:

$$100\,000 * 1.03 = 103\,000 \quad kr.$$

Hvor lang tid det tar for at hun skal ha 300 000 kr. på kontoen sin, kan vi beregne ved å bruke en *while*-løkke. Løkken looper helt til vi når sluttbeløpet. For å løse oppgaven lager vi en funksjon som tar startbeløp, rentefot og sluttbeløp som parametere, og returnerer saldo og antall år.

Algoritme

1 Definere en funksjon *beregn_tid_gitt_innskudd()* som tar startbeløp, rentefot, og sluttbeløp som parametere.
   1.1 Definere vekstfaktoren gitt ved $1 + rentefoten/100$.
   1.2 Definere en variabel saldo; Saldoen i starten er lik startbeløp.
   1.3 Definere en variabel for å telle antall år, og sette den lik null.
   1.4 Bruke en *while*-løkke som looper så lenge saldoen er mindre enn sluttbeløp.
      1.4.1 Beregne saldoen, og sette ny verdi for den.
      1.4.2 Inkrementere antall år med 1.
   1.5 Returnere antall år og saldoen.
2 Lage en funksjon *main()* som klientprogram.
   2.1 Skrive ut funksjonaliteten til programmet for brukere (valgfri).
   2.2 Få startbeløp, rentefot og sluttbeløp fra brukeren, og konvertere verdiene til flyttall.
   2.3 Beregne antall år og saldoen ved å kalle *beregn_tid_gitt_innskudd()*-funksjonen.
   2.4 Skrive ut resultatet.
3 Kalle *main()*-funksjonen.

```python
# a

'''
funksjon for å beregne tid for en sluttverdi med innskudd
funksjonen tar startbeløp, rentefot, og sluttbeløp som parametere
funksjonen returnerer antall år, og saldoen
'''
def beregn_tid_gitt_innskudd(startbeløp, rentefoten, sluttbeløp):

    # definerer vekstfaktorvariabelen
    vekstfaktor = 1 + rentefoten/100

    # definerer en variabel saldo og setter den lik startbeløpet
    saldo = startbeløp

    # definerer en variabel år
    antall_år = 0

    # løkken looper frem til saldoen er >= sluttbeløp
    while saldo < sluttbeløp:

        # saldoen øker hvert år med vekstfaktoren
        saldo *= vekstfaktor

        # etter hver loop inkrementerer antall år med 1
        antall_år += 1

    # returnerer resultatet
    return antall_år, saldo


def main():

    print('Oppgave a: Antall år for å spare et sluttbeløp med innskudd')

    # får verdier fra brukeren
    startbeløp = float(input('Oppgi startbeløpet: '))
    sluttbeløp = float(input('Oppgi sluttbeløpet: '))
    rentefoten = float(input('Oppgi rentefoten: '))

    # kaller metoden beregn_tid_gitt_innskudd() for å beregne tiden
    antall_år, saldo = beregn_tid_gitt_innskudd(startbeløp, rentefoten, sluttbeløp)

    # skriver ut resultatet
    print(f'\nDet tar {antall_år} år for å ha {sluttbeløp} kr. på kontoen.\
    \nSaldoen skal være {round(saldo)} kr.')

main()
```

Program A.26: Programmet beregner tiden for en sluttverdi med innskudd.

Output:

```
Oppgave a: Antall år for å spare et sluttbeløp med innskudd
Oppgi startbeløpet: 100000
Oppgi sluttbeløpet: 300000
Oppgi rentefoten: 3

Det tar 38 år for å ha 300000.0 kr. på kontoen.
Saldoen skal være 307478 kr.
```

b Vi kan modifisere funksjonen ved å legge til en parameter kalt *utbetaling*, og summere den med saldoen i hver loop. Algoritmen for programmering er derfor den samme som forrige oppgave.

Algoritme

1 Definere en funksjon *beregn_tid_gitt_innskudd_utbetaling()* som tar startbeløp, rentefot, sluttbeløp og utbetaling som parametere.
  1.1 Definere vekstfaktoren gitt ved $1 + rentefoten/100$.
  1.2 Definere en variabel saldo; Saldoen i starten er lik startbeløp.
  1.3 Definere en variabel for å telle antall år, og sette variabelen lik 0.
  1.4 Bruke en *while*-løkke som looper så lenge saldoen er mindre enn sluttbeløp.
    1.4.1 Beregne saldoen, og sette ny verdi for den.
    1.4.2 Inkrementere antall år med 1.
  1.5 Returnere antall år og saldoen.
2 Definere *main()*-funksjon som klientprogram.
  2.1 Skrive ut funksjonaliteten til programmet for brukere (valgfri).
  2.2 Få startbeløp, rentefot, sluttbeløp og utbetaling fra brukeren, og konvertere verdiene til flyttall.
  2.3 Beregne antall år og saldoen ved å kalle *beregn_tid_gitt_innskudd_utbetaling()*-funksjonen.
  2.4 Skrive ut resultatet.
3 Kalle *main()*-funksjonen.

```python
# b

'''
funksjon for å beregne tid for en sluttverdi med innskudd og årlig utbetaling
funksjonen tar startbeløp, rentefot, sluttbeløp og utbetaling som parametere
funksjonen returnerer antall år, og saldoen
'''
def beregn_tid_gitt_innskudd_utbetaling(startbeløp, rentefoten, sluttbeløp, utbetaling):

    # definerer vekstfaktorvariabelen
    vekstfaktor = 1 + rentefoten/100

    # definerer en variabel saldo og setter den lik startbeløpet
    saldo = startbeløp

    # definerer en variabel år
    antall_år = 0

    # løkken looper frem til saldoen er >= sluttbeløp
    while saldo < sluttbeløp:

        # saldoen øker hvert år med vekstfaktoren, utbetalingen legges til
        saldo = saldo * vekstfaktor + utbetaling

        # etter hver loop inkrementerer antall år med 1
        antall_år += 1

    # returnerer resultatet
    return antall_år, saldo

def main():

    print('\nOppgave b: Antall år for å spare et sluttbeløp med innskudd og årlig utbetaling')

    # får utbetaling fra brukeren
    startbeløp = float(input('Oppgi startbeløpet: '))
    sluttbeløp = float(input('Oppgi sluttbeløpet: '))
    rentefoten = float(input('Oppgi rentefoten: '))
    utbetaling = float(input('Oppgi utbetalingsbeløpet: '))

    # kaller metoden beregn_tid_gitt_innskudd_utbetaling() for å beregne tiden
    antall_år_2, saldo_2 = beregn_tid_gitt_innskudd_utbetaling(startbeløp, rentefoten,
                                                               sluttbeløp, utbetaling)

    # skriver ut resultatet
    print(f'\nDet tar {antall_år_2} år for å ha {sluttbeløp} kr. på kontoen.\
    \nSaldoen skal være {round(saldo_2)} kr.')

main()
```

Program A.27: Programmet beregner tiden for en sluttverdi med innskudd og årlig utbetaling.

Output:

```
Oppgave b: Antall år for å spare et sluttbeløp med innskudd og årlig utbetaling
Oppgi startbeløpet: 100000
Oppgi sluttbeløpet: 300000
Oppgi rentefoten: 3
Oppgi utbetalingsbeløpet: 15000

Det tar 10 år for å ha 300000.0 kr. på kontoen.
Saldoen skal være 306350 kr.
```

6 Matematikk R1

Kompetansemål etter matematikk R1

Mål for opplæringen er at eleven skal kunne

- bestemme den deriverte i et punkt geometrisk, algebraisk og ved numeriske metoder, og gi eksempler på funksjoner som ikke er deriverbare i gitte punkter
- modellere og analysere eksponentiell og logistisk vekst i reelle datasett
- forstå begrepet vektor og regneregler for vektorer i planet, og bruke vektorer til å beregne ulike størrelser i planet

Kilde

6.1 Numeriske metoder

Oppgave Undersøk om det er halverings metode eller Newton-Raphsons metode som bruker kortest tid på å løse en likning $f(x)$ med et avvikskrav på 10^{-5}. Bruk *time()*-funksjonen for å måle tiden.

Oppgaven er hentet fra (Bueie, 2019, s. 138).

Løsning Newton-Raphsons metode A.43 tar utgangspunkt i en initial gjetning x_0 for en funksjon $f(x)$ for å finne $f(x+1)$. Vi gjentar metoden helt til $f(x+1) = 0$.

Halverings metode A.42 er basert på å halvere et intervall $[a, b]$ til å finne roten til en likning $f(x)$, vi fortsetter halveringen til vi finner en verdi x hvor $f(x) = 0$.

Kildekoden for Newton-Raphsons metode og halverings metode er lånet fra GeeksforGeeks (2021a, 2021b).

Vi løser oppgaven ved å ta i bruk de to metodene for å finne løsningen for likningen $f(x) = x^3 - x^2 + 2$, mens vi måler kjøretiden for de to metodene.

Algoritme for Newton-Raphsons metode

1 Definere *Newton_Raphsons_metode()*-funksjonen; funksjonen tar x som parameter.

- 1.1 Definerer en variabel *iterasjon* for å telle antall iterasjoner, og sette den lik 0.

  Vi teller iterasjoner for hver metode og til slutt sammenligner vi iterasjonene for de to metodene.
- 1.2 Beregne $\frac{f(x)}{f'(x)}$ og lagre verdien i en variabel kalt h.
- 1.3 Bruke en *while*-løkke og loope så lenge h er større eller lik toleransen for feil (avvikskrav).
  - 1.3.1 Beregne h ved å dele $f(x)$ på $f'(x)$.
  - 1.3.2 Beregne verdien til x ved å trekke h fra x. (Se formelen for Newton-Raphsons metode A.43).
  - 1.3.3 Inkrementere iterasjoner med 1.
- 1.4 Returnere x-verdien og antall iterasjoner.

```
'''
Programmet implementerer Newton-Raphsons metode,
kildekoden for er lånet fra GeeksforGeeks.
'''

def Newton_Raphsons_metode(x):

    # definerer en variabel iterasjon for å telle iterasjoner
    iterasjon = 0

    # definerer en verdi h som er lik f/f'
    h = f(x) / f_derivert(x)

    # bruker while-løkke for å finne nullpunktet, løkken looper så lenge h >= toleranse
    while abs(h) >= toleranse:

        h = f(x) / f_derivert(x)

        # Newton-Raphsons metode -> x(i+1) = x(i) - f(x) / f'(x)
        x = x - h

        # inkrementerer iterasjoner med 1
        iterasjon += 1

    # hvis verdien er funnet, skriver ut resultatet
    # print(f'Løsningen av f(x) er {x:.5f}')

    # returnerer resultatet
    return x, iterasjon
```

Program A.28: Programmet implementerer Newton-Raphsons metoden.

Algoritme for halverings metode

1 Definere *halverings_metode()*-funksjonen; funksjonen tar a og b som parametere.

- 1.1 Definerer en variabel *iterasjon* for å telle opp iterasjoner.
- 1.2 Sjekke om $f(a) * f(b) >= 0$;
  - 1.2.1 hvis ja, informerer brukeren om at løsningen ikke finnes på intervallet, og returnerer.
- 1.3 Definere en variabel m som står for *midtpunkt*, og sette den lik a.
- 1.4 Bruke en *while*-løkke, og loope så lenge $(b - a)$ (rekkevidde til intervallet) er større eller lik toleransen.
  - 1.4.1 Finne midtpunktet m på intervallet $[a, b]$.
  - 1.4.2 Sjekke om $f(m)$ er lik 0;
    - 1.4.2.1 hvis ja, m er løsningen av $f(x)$, og vi slutter løkken med *break*.
  - 1.4.3 Sjekke om $f(a) * f(m)$ er mindre enn 0;
    - 1.4.3.1 hvis ja, det betyr at $f(a)$ og $f(m)$ ligger på forskjellige sider av x-aksen, altså ligger nullpunktet mellom punktene a og m. Sette $b = m$, dvs. vi lager et nytt intervall.
  - 1.4.4 Ellers
    - 1.4.4.1 setter vi $a = m$. Det betyr at nullpunktet ligger mellom punktene m og b.
  - 1.4.5 Inkrementere iterasjoner med 1.
- 1.5 Returnere m og antall iterasjoner.

```
'''
Programmet implementerer halverings metode,
kildekoden er lånet fra GeeksforGeeks.
'''

def halverings_metode(a, b):

    # definerer en variabel iterasjon for å telle iterasjoner
    iterasjon = 0

    # hvis f(a) og f(b) ligger på samme side av x-aksen
    if (f(a) * f(b) >= 0):

        # skrive ut info til brukeren
        print('Du har ikke valgt riktig intervall')

        # og returnerer
        return

    # setter midtpunktet lik startpunkt på intervallet
    m = a

    # bruker while-løkke, og går i løkken så lenge lengden av intervallet er >= toleranse
    while ((b-a) >= toleranse):

        # finner midtpunkt av intervallet
        m = (a+b) / 2

        # sjekker om f(m) = 0, altså hvis m er nullpunktet
        if (f(m) == 0.0):

            # slutter løkken
            break

        # hvis f(m) og f(a) ligger på forskjellige sider av x-aksen
        if (f(m)*f(a) < 0):

            # setter b=m
            b = m

        # ellers setter a=m
        else:
            a = m

        # inkrementerer iterasjoner med 1
        iterasjon += 1

    # dersom f(m)=0, skriver ut resultatet
    # print(f'Løsningen av f(x) er {m:.5f}')

    # returnerer resultatet
    return m, iterasjon
```

Program A.29: Programmet implementerer halverings metoden.

Algoritme for å løse oppgaven

1 Importere modulen *time*.

2 Definere avvikskrav som konstant.

3 Definere en funksjon *f()* som returnerer funksjonen $f(x) = x^3 - x^2 + 2$; funksjonen tar x som parameter.

4 Definere en funksjon *f_derivert()* som returnerer den deriverte til $f(x)$; funksjonen tar x som parameter.

5 Definere funksjonen *main()* som klientprogram.

5.1 Definere x_0, a og b som variabler, og sette verdi for dem.

5.2 Sette starttidspunkt for kjøring av Newton-Raphsons metoden ved å kalle *time()*-funksjonen fra *time*-modulen.

5.3 Finne løsningen til likningen $f(x)$ for x_0, samt antall iterasjoner ved å kalle *Newton_Raphsons_metode()*-funksjonen.

5.4 Skrive ut resultatet (svar, tid, og iterasjoner).

5.5 Sette starttidspunkt for kjøring av halverings metoden ved å kalle *time()*-funksjonen fra *time*-modulen.

5.6 Finne løsningen til likningen $f(x)$ for x_0, samt antall iterasjoner ved å kalle *halverings_metode()*-funksjonen.

5.7 Skrive ut resultatet (svar, tid, og iterasjoner).

6 Kalle *main()*-funksjonen.

```
'''
programmet sammenligner kjøretiden av Newton-Raphsons metode og Halverings metode
programmet finner løsning for f(x)=x^3 - x^2 + 2
'''

# importerer modulen time
import time

# definerer toleranse som konstant
toleranse = 0.00001

# definerer en funksjon f som returnerer funksjonen vår
def f( x ):
    return x**3 - x**2 + 2

# definerer en funksjon f_derivert som returnerer den deriverte av f
def f_derivert( x ):
    return 3*x**2 - 2*x

# metoden for å beregne kjøretiden for de to metodene
def main():

    # setter startverdier
    x_0 = -20
    a = -200
    b = 300

    # tester Newton-Raphsons metoden
    starttid_1 = time.time()
    resultat_1, iterasjon_1 = newton_Raphsons_metode(x_0)

    print(f'Løsningen av f(x) er {resultat_1:.5f}, \
    kjøre tiden for Newton-Raphsons metode er \
    {time.time() - starttid_1} med {iterasjon_1} iterasjoner')

    # tester Halverings metoden
    starttid_2 = time.time()
    resultat_2, iterasjon_2 = halverings_metode(a, b)

    print(f'Løsningen av f(x) er {resultat_2:.5f},\
    kjøre tiden for Halverings metode er\
    {time.time() - starttid_2} med {iterasjon_2} iterasjoner')

main()
```

Program A.30: Programmet sammenligner kjøretiden av Newton-Raphsons metode og halverings metode.

Eksempel output:

```
Løsningen av f(x) er -1.00000, kjøre tiden for Newton-Raphsons metode er 0.0 med 11 iterasjoner
Løsningen av f(x) er -1.00001, kjøre tiden for Halverings metode er 0.001001596450805664 med 26 iterasjoner
```

Vi får forskjellige verdier for tid og iterasjoner avhengig av hvilke startverdier vi velger. Kjøretiden for begge metodene er veldig kort slik at vi får 0 som kjøretid i de fleste av kjøringene. I eksempeloutputen over fikk vi tall for halverings metoden, men fortsatt 0 for Newton-Raphsons metoden. Det betyr at kjøretiden for den sist nevnte er kortere, noe som vi kan betrakte fra antall iterasjoner for de to metodene også. I alle kjøringene av programmet får vi flere iterasjoner for halverings metoden enn for Newton-Raphsons metoden.

Vi kan derfor konkludere at kjøretiden for Newton-Raphsons metoden er kortest.

6.2 Derivasjon

Derivasjon er blant de matematiske begrepene som har flere regler og teknikker. Dette kan føre til at elever blir distrahert fra å forstå hva derivasjon faktisk er. Som løsning for dette kan numerisk derivasjon gi en bedre forståelse av hva vi gjør når vi deriverer en funksjon, i og med at numerisk derivasjon viser faktisk definisjonen for den deriverte (Haraldsrud mfl., 2020, s. 211).

Vi beregner den deriverte av en funksjon analytisk gitt ved formelen:

$$f'(x) = \lim_{\Delta x \to 0} = \frac{f(x + \Delta x) - f(x)}{\Delta x} \tag{A.14}$$

Vi kan beregne en tilnærming for denne grensen der Δx går mot 0 med en svært liten Δx. Denne metoden kalles *Newtons kvotient* (Haraldsrud mfl., 2020, s. 211):

$$f'(x) \approx \frac{f(x + \Delta x) - f(x)}{\Delta x} \tag{A.15}$$

Oppgave Finn den deriverte til funksjonen $f(x) = \sqrt{x}$ i punkt $x = 3$.

Løsning Bueie (2019, s. 100) bruker følgende kode for å finne den deriverte til en gitt funksjon i et gitt punkt:

```
'''
Programmet finner den deriverte til f(x) numerisk i punkt x=3
'''

# definerer en funksjon f som returnerer vår funksjon
def f(x):
    return np.sqrt(x)

# definerer en funksjon som returnerer den deriverte av f
def derivert(x):

    # definerer steglengde
    delta_x = 0.001

    # finner endringer i y-verdier
    endring = f(x + delta_x) - f(x)

    # finner veksthastigheten ved å dele endringen i y på endringen i x
    vekst = endring/delta_x

    # returnerer veksthastigheten
    return vekst

# finner den deriverte av f in punkt x=3 med en steglengde=0.001
print(f'Den deriverte av funksjonen f i punkt x = 3 er lik {derivert(3):.3f}')
```

Program A.31: Programmet finner den deriverte til $f(x)$ numerisk i punkt $x = 3$.

Output:

```
Den deriverte av funksjonen f i punkt x = 3 er lik 0.289
```

Vi kan utvide koden ved å legge til tegning av grafene $f(x)$ og $f'(x)$ samt å spørre brukeren om i hvilket punkt og med hvilken steglengde programmet skal finne den deriverte av f.

Algoritme

1 Importere *NumPy*-biblioteket, og *matplotlib.pyplot*-modulen.

2 Definere en funksjon som returnerer funksjonen $f(x) = \sqrt{x}$; funksjonen tar x som parameter.

3 Definere en funksjon kalt *derivert* som tar x og Δx som parametere.

3.1 Benytte formelen A.15 for å finne den deriverte av $f()$. Her kan vi enten definere en variabel *vekst* (Du kan kalle variabelen hva du ønsker) og finne den deriverte med en gang,

```
vekst = (f(x + delta_x) - f(x)) / delta_x
```

eller vi kan finne først telleren av formelen og så dele den på nevneren.

```
endring = f(x + delta_x) - f(x)
vekst = endring/delta_x
```

3.2 Returnere vekst.

4 Definere funksjonen *main()* som klientprogram.

4.1 Få x-verdien fra brukeren (Hvilket punkt programmet skal finne den deriverte for); konvertere input-verdien til heltall.

4.2 Spørre brukeren om steglengde. Denne størrelsen må defineres som *float* fordi som regel er det ønskelig å defineres små verdier for Δx.

4.3 Beregne den deriverte av $f(x)$ ved å kalle *derivert()*-funksjonen.

4.4 Skrive ut resultatet.

4.5 Spørre brukeren om start- og sluttpunkt på x-aksen samt antall punkter for tegning av grafer. Denne verdien er som standard lik 50. Konvertere input-verdiene til heltall.

4.6 Definere x-verdier ved å kalle *linspace()*-funksjonen fra *NumPy*-biblioteket. Sette de verdiene som er mottatt fra brukeren som parameter inn i funksjonen.

4.7 Definere y-verdier for $f()$-funksjonen. Funksjonen tar *x-verdier* som parameter.

4.8 Definere y-verdier for den deriverte av $f()$, $(f'(x))$ ved å kalle *derivert()*-funksjonen. Funksjonen tar *x-verdier* og *delta_x* som parametere.

4.9 Sette størrelse for plottet ved å kalle *figure()*-funksjonen fra *pyplot*. Vi definerer figurstørrelse som parameter til funksjonen:

```
figure(figsize=(x,y))
```

4.10 Sette tittel for plottet ved å kalle *title()*-funksjonen fra *pyplot*.

4.11 Sette label for x- og y-aksen ved å kalle henholdsvis *xlabel()*- og *ylabel()*-funksjonen.

4.12 Kalle *plot()*-funksjonen for å tegne grafer for $f(x)$ og $f'(x)$. Vi kan kalle funksjonen to ganger og hver gang sette inn x- og y-verdier for funksjonen, eller vi kan kalle *plot()*-funksjonen én gang og sette x- og y-verdier for begge funksjoner etter hverandre. Det er også mulig å definere forskjellige farger for grafer ved å skrive fargekode som streng etter x og y:

```
plot(x_verdier, y_verdier,'r')
```

her står 'r' for rødfarge.

For å vise punktet som brukeren ønsker å finne den derivert i, setter vi x og y-derivert inn i *plot()*-funksjonen. For punktet bruker vi 'o' eller '*farge* o' som parameter etter x- og y-verdi.

4.13 Vise koordinater for det deriverte punktet på plottet ved hjelp av *text()*-funksjonen fra *pyplot*-modulen. Funksjonen tar x- og y-verdi og teksten som parametere. Syntaksen for å skrive koordinater som tekst er som følger:

```
text(x, y,'({},␣{})'.format(x, y))
```

4.14 Kalle *legend()*-funksjonen og skrive navnene til funksjoner (som en liste av strenger) som parameter.

4.15 Sette rutenett på plottet ved å kalle *grid()*-funksjonen.

4.16 Vise plottet ved å kalle *show()*-funksjonen.

5 Kalle *main()*-funksjonen.

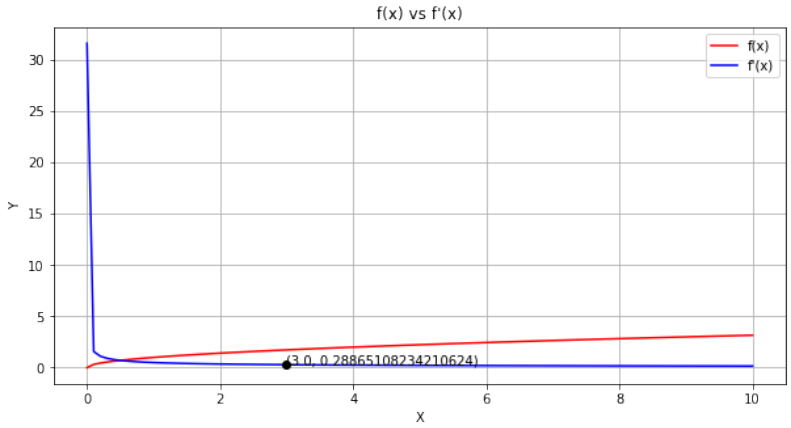
```python
'''
utvidet kode for numerisk derivasjon, programmet tegner grafen til f(x) og f'(x),
og finner den derivert til f(x) i x=x_i grafisk
'''

# importerer biblioteker
import matplotlib.pyplot as plt
import numpy as np

# definerer en funksjon f som returnerer vår funksjon
def f(x):
    return np.sqrt(x)

# definerer en funksjon som returnerer den deriverte av f
def derivert(x, delta_x):

    endring = f(x + delta_x) - f(x) # finner endringer i y-verdier
    vekst = endring/delta_x   # finner veksthastigheten ved å dele endringen i y på endringen i x

    return vekst  # returnerer veksthastigheten

# funksjonen for å teste metoden samt tegning av grafer
def main():
    '''
    data for beregning av f'(x)
    '''
    # får x-verdien og steglengde fra brukeren
    x = float(input('Skriv inn x-verdien: '))
    delta_x = float(input('Skriv inn steglengde: '))

    # kaller derivert()-metoden for å finne den deriverte av f
    f_derivert = derivert(x, delta_x)

    # skrive ut resultatet
    print(f'\nDen deriverte av funksjonen f i punkt x = {x} er lik {f_derivert:.3f}\n')

    '''
    data for tegning av grafer
    '''
    # får startpunkt, sluttpunkt og antall punkter for x-aksen fra brukeren
    x_start = int(input('Skriv inn startverdi: '))
    x_slutt = int(input('Skriv inn sluttverdi: '))
    antall_punkter = int(input('Skriv inn antall sampler: '))

    # definerer x-verdier for grafen
    x_verdier = np.linspace(x_start, x_slutt, antall_punkter)

    # definerer y-verdier for f(x) og f'(x)
    y_verdier = f(x_verdier)
    y_derivert = derivert(x_verdier, delta_x)

    plt.figure(figsize=(10,5))   # setter størrelse for plottet

    # setter tittel for plottet, og label for aksene
    plt.title('f(x) vs f\'(x)')
    plt.xlabel('X')
    plt.ylabel('Y')

    # tegner grafen for f(x) og f'(x)
    plt.plot(x_verdier, y_verdier, 'r', x_verdier, y_derivert, 'b', x, f_derivert, 'ko')
    # viser koordinat for derivert punkt på plottet
    plt.text(x, f_derivert, '({}, {})'.format(x, f_derivert))

    plt.legend(['f(x)', 'f\'(x)'])  # legger til label for funksjoner
    plt.grid()  # setter rutenett på plottet
    plt.show()  # viser plottet

main()
```

Program A.32: Programmet tegner grafen til $f(x)$ og $f'(x)$, og finner den derivert til $f(x)$ i $x = x_i$ grafisk.

Output:

```
Programmet finner den deriverte av f(x) = -x

Skriv inn x-verdien: 3
Skriv inn steglengde: .001

Den deriverte av funksjonen f i punkt x = 3.0 er lik 0.289

Nå tegner grafen til f(x) og f'(x)

Skriv inn startverdi: 0
Skriv inn sluttverdi: 10
Skriv inn antall sampler: 100
```

Tilnærmet verdi for den deriverte av $f(x) = \sqrt{x}$ i punkt $x = 3$, med tre desimaler nøyaktighet er 0.289. Figur A.12 viser også den samme løsningen grafisk.



Figur A.12: Grafisk løsning av $f(x) = \sqrt{x}$ i punkt $x = 3$.

6.3 Vektorregning

"En størrelse som har en bestemt lengde og en bestemt retning kalles en *vektor*. Eksempler på vektorer er forflytning, fart og kraft.

En skalar er en størrelse uten retning. Eksempler på skalarer er temperatur, areal og volum"(Kristensen & Aanensen, 2018f).

Summen av to vektorer $\vec{v}$ og $\vec{u}$ er en ny vektor lik:

$$\vec{v} + \vec{u} = [v_1 + u_1, v_2 + u_2, v_3 + u_3] \tag{A.16}$$

Skalarproduktet mellom to vektorer $\vec{v}$ og $\vec{u}$ er et reelt tall definert ved:

$$\vec{v}.\vec{u} = |\vec{v}|.|\vec{u}|.\cos\theta = v_1.u_1 + v_2.u_2 + v_3.u_3 \tag{A.17}$$

her $|\vec{v}|$ er lik størrelse til lengden av $\vec{v}$, og θ er vinkelen mellom de to vektorene.

Skalarproduktet er null dersom de to vektorene står vinkelrett på hverandre, og vektorene sies da å være *ortogonale*.

Vi kan skrive lengden av en vektor som:

$$|\vec{v}| = \sqrt{\vec{v}.\vec{v}} = \sqrt{x^2 + y^2 + z^2} \tag{A.18}$$

Oppgave Lag et program som tar inn to vektorer, og bruker arealsetningen til å finne arealet til trekanten utspent av de to vektorene.

Hint: Dere vil trenge å lage en funksjon som regner ut lengden til en vektor, og en funksjon som finner vinkel mellom de to vektorene.

Oppgaven er hentet fra (Haraldsrud mfl., 2020).

Løsning Arealsetning brukes når vi vil finne areal til en vilkårlig trekant når to av sidelengdene og vinkelen mellom dem er kjent. Formelen for arealsetningen er gitt ved:

$$A = \frac{1}{2} a.h. \sin\theta \tag{A.19}$$

her a er lengden til grunnlinjen i trekanten og h er hypotenusen (Matematikk.org, udatert).

Vi bruker skalarproduktet for å finne vinkelen mellom de to vektorene. Omformulering av skalarproduktet med hensyn til vinkelen θ gir:

$$\theta = \cos^{-1}\left(\frac{\vec{v}.\vec{u}}{|\vec{v}|.|\vec{u}|}\right) \tag{A.20}$$

For å løse oppgaven på en strukturert måte, velger vi å opprette en *class* for *vektor*. Det betyr at vi lager et objekt av vektor som har egne egenskaper og metoder.

Algoritme

1 Importere modulen *math*.

2 Opprette en klasse *Vektor*.

2.1 Bruke *__init__()*-funksjonen for å definere konstruktøren til *Vektor* [11].
For klassen *Vektor* setter vi x, y, og z koordinater som verdier til en instans av vektor-objektet. Disse verdiene defineres som parametere til Vektor.

2.2 Definere *lengde* som en egenskap av *Vektor*. Vi bruker formelen A.18 for lengde til en vektor for å beregne lengden.

3 Definere en *dot_produkt()*-funksjon som beregner produktet av to vektorer. Funksjonen tar to vektor-objekter som parametere.

[11] Alle klasser har en funksjon kalt *__init__ ()*, som alltid genereres ved oppretting av en klasse. Vi bruker *__init__()*-funksjonen til å tilordne verdier til objektets egenskaper (W3Schools, udatert-a).

3.1 Beregne produktet av de to vektorene ved hjelp av formelen A.17, og lagre verdien i en variabel kalt *produkt*.

3.2 Returnere *produkt*.

4 Definere en funksjon kalt *vektor_vinkel()* for å finne vinkelen mellom to vektorer. Funksjonen tar to vektorer som parametere.

4.1 Beregne vinkelen mellom de to vektorene ved hjelp av formelen A.20, og lagre verdien i en variabel kalt *theta*.

4.2 Returnere *theta*.

5 Definere en funksjon kalt *areal_utspent()*; funksjonen tar to vektor-objekter som parametere og beregner arealet til trekanten utspent av de to vektorene.

5.1 Beregne grunnlinjen, og hypotenusen til trekanten. For dette finner vi lengde til vektorene $\vec{v}$ og $\vec{u}$ ved å bruke *v.lengde* og *u.lengde*.

5.2 Finne vinkelen mellom de to vektorene ved å kalle *vektor_vinkel()*-funksjonen.

5.3 Bruke arealsetningen A.19 for å finne arealet til trekanten utspent av de to vektorene, og returnere resultatet.

6 Definere en funksjon *main()* som klientprogram.

6.1 Få x, y og z-verdiene for to vektorer $\vec{v}$ og $\vec{u}$ fra brukeren; konvertere alle input-verdiene til flyttall.

6.2 Opprette vektorene $\vec{v}$ og $\vec{u}$ ved å bruke *Vektor*-klassen.

6.3 Finne arealet til trekanten utspent av de to vektorene ved å kalle *areal_utspent()*-funksjonen.

6.4 Skrive ut resultatet.

7 Kalle *main()*-funksjonen.

```python
'''
Programmet finner areal av trekanten utspent av to vektorer v og u
'''
# importerer modulen math
import math

# oppretter en klasse for vektorobjekt
class Vektor:

    '''
    konstruktør for vektor-objektet
    verdier til vektorobjektet: x, y og z koordinater

    Self-parameteren er en referanse til foreløpig instans av klassen,
    og brukes til å få tilgang til variabler som tilhører klassen.
    Du kan endre self til et hva som helst navn, men den må være den første parameteren i alle metoder
    '''
    def __init__(self, x, y, z):

        # definerer x, y og z-koordinater til en vektor
        self.x = x
        self.y = y
        self.z = z

        # definerer lengden av en vektor
        self.lengde = math.sqrt(self.x**2 + self.y**2 + self.z**2)

# metoden for å finne dot-produkt av to vektorer
def dot_produkt(v, u):

    # beregner produktet
    produkt = v.x * u.x + v.y * u.y + v.z * u.z

    # returnere resultatet
    return produkt

# funksjon for å finne vinkelen mellom to vektorer
def vektor_vinkel(v, u):

    # bruker omformulert skalarprodukt-formelen til å finne vinkel mellom v og u
    theta = math.acos(dot_produkt(v, u) / (v.lengde * u.lengde))

    # returnerer resultatet
    return theta

# funksjon for å finne arealet til trekanten utspent av to vektorer
def areal_utspent(v, u):

    # definerer grunnlinje til utspent trekant
    grunnlinje = v.lengde

    # definerer hypotenus til trekanten
    hypotenus = u.lengde

    # finner vinkelen mellom grunnlinjen og hypotenusen
    theta = vektor_vinkel(v, u)

    # beregner arealet ved hjelp av arealsetning
    areal = 0.5 * grunnlinje * hypotenus * math.sin(theta)

    # returnerer resultatet
    return areal
```

Program A.33: Programmet finner areal av trekanten utspent av to vektorer $\vec{v}$ og $\vec{u}$.

.

```
# klientprogram
def main():

    # får verdier fra brukeren for vektor v
    x_1 = float(input("Oppgi x-verdien til vektoren v: "))
    y_1 = float(input("Oppgi y-verdien til vektoren v: "))
    z_1 = float(input("Oppgi z-verdien til vektoren v: "))

    # får verdier fra brukeren for vektor u
    x_2 = float(input("Oppgi x-verdien til vektoren u: "))
    y_2 = float(input("Oppgi y-verdien til vektoren u: "))
    z_2 = float(input("Oppgi z-verdien til vektoren u: "))

    # lager vektor v
    v = Vektor(x_1, y_1, z_1)

    # lager vektor u
    u = Vektor(x_2, y_2, z_2)

    # beregner arealet til trekanten utspent av de to vektorene
    areal = areal_utspent(v, u)

    # skriver ut resultatet
    print(f'Arealet til trekanten utspent av vektorene v og u er {areal:.2f}')

main()
```

Program A.34: Klientprogram for å finne areal av trekanten utspent av to vektorer $\vec{v}$ og $\vec{u}$

Eksempel output:

```
Oppgi x-verdien til vektoren v: 1
Oppgi y-verdien til vektoren v: 2
Oppgi z-verdien til vektoren v: 3
Oppgi x-verdien til vektoren u: 3
Oppgi y-verdien til vektoren u: 2
Oppgi z-verdien til vektoren u: 1
Arealet til trekanten utspent av vektorene v og u er 4.90
```

6.4 Regresjonsanalyse

Regresjonsanalyse handler om å finne ut hvilken sammenheng det finnes mellom to fenomener. For eksempel å finne ut hvilken sammenheng det finnes mellom snorlengden og svingetiden til en pendel. Vi kan studere en regresjonsmodell enten som en *lineær regresjon* eller som en *ikke-lineær regresjon*.

Oppgave Tabellen viser utslippene av karbondioksid CO_2 i verden målt i millioner tonn.

Årstall	1980	1990	2000	2005	2006
Utslipp av CO2 i millioner tonn	18 054	20 988	23 509	27 146	28 003

Plott punktene i tabellen i et koordinatsystem, og finn en matematisk modell som beskriver utslippene av CO_2. La x være antall år etter 1980 og $U(x)$ utslippene av CO_2.

Mange land har vedtatt å senke utslippet av CO_2 i tiden framover. Vurder gyldigheten framover i tid av modellen du fant.

Oppgaven er hentet fra (Kristensen & Aanensen, 2018a).

Løsning Vi finner både lineære og ikke-lineære regresjonsmodeller, og så sammenligner vi modellene for å vise hvilken regresjonsmodell gir bedre nøyaktighet.

Vi setter først dataene i form av punkter i koordinatsystemet hvor x-verdier skal være antall år etter 1980, og y-verdier utslippene av CO_2. Vi definerer dermed følgende liste som x-verdier:

[0, 10, 20, 25, 26]

Vi skal finne eksponential regresjon, lineær regresjon og polynomisk regresjon av grad $2, 3, ...$, og konkludere til slutt hvilken modell tilpasser best dataene.

Eksponential regresjon

Algoritme

1 Importere *Numpy*-biblioteket, modulen *matplotlib.pyplot*, og funksjonen *curve_fit()* fra *scipy.optimize*-modulen.

2 Definere x- og y-verdier som *NumPy-array*er.

3 Definerer en funksjon kalt *eksp_funksjon()* som returnerer en eksponential funksjon på formen $a.e^{b.x}$ [12]; funksjonen tar to konstanter a og b, og x som parametere.

4 Benytte funksjonen *curve_fit()* for å finne koeffisienter til regresjonsfunksjonen.

Funksjonen tar følgende parametere (funksjonen tar flere parametere, men vi går bare gjennom de som blir brukt i programmet):

- **f:** Modellens funksjon
- **xdata:** x-verdier i datasettet
- **ydata:** y-verdier i datasettet

her modellens funksjon er vår eksponential funksjon.

5 Sette koeffisientene samt x-verdiene i *eksp_funksjon()*, og lage en ny liste av y-verdier. Vi trenger disse verdiene for å plotte regresjonslinjen.

6 Sette tittel for plottet ved å kalle *title()*-funksjon fra *matplotlib.pyplot*-modulen.

7 Sette label for x- og y-aksen ved å kalle henholdsvis *xlabel()*- og *ylabel()*-funksjon fra *matplotlib.pyplot*-modulen.

8 Bruke *scatter()*-funksjonen for å sette dataene i koordinatsystemet.

scatter() tar x- og y-verdiene (koordinat) som parametere, og finner punkter med gitte x- og y-koordinater.

9 Bruke *plot()*-funksjonen for å plotte regresjonslinjen. Funksjonen tar x- og y-verdier som parametere.

Det er også mulig å sette farge for grafen ved å legge for eks. 'r' som parameter i funksjonen. 'r' står for rødfarge.

10 Benytte *show()*-funksjonen for å vise plottet.

11 Skrive ut funksjonen til regresjonslinjen.

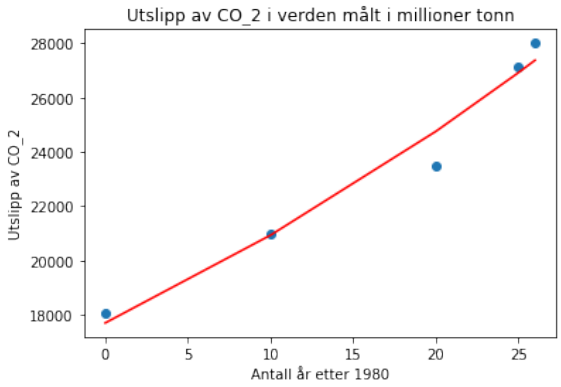
[12] I programmet har vi brukt *lambda*-funksjon i stedet for å definere en egen funksjon. En *lambda*-funksjon er en liten anonym funksjon som kan ta et hvilket som helst antall argumenter, men kan bare ha ett uttrykk (W3Schools, udatert-b).

```
'''
program for å finne eksponential regresjon for CO_2 utslipp i verden
'''

# importerer biblioteker
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

# definerer x-verdier
x = np.array([0, 10, 20, 25, 26])

# definerer y-verdier
y = np.array([18054, 20988, 23509, 27146, 28003])

# definerer eksponential funksjonen på formen y = a.e^b.x
eksp_funksjon = lambda x1, a, b: a * np.exp(b * x1)

# kaller på funksjonen curve_fit() for å finne koeffisienter til regresjonsfunksjonen
[a, b], res1 = curve_fit(eksp_funksjon, x,  y)

# beregner y-verdier til regresjonsfunksjonen
y_verdier = eksp_funksjon(x, a, b)

# setter tittel, x- og y-label
plt.title('Utslipp av CO_2 i verden målt i millioner tonn')
plt.xlabel('Antall år etter 1980')
plt.ylabel('Utslipp av CO_2')

# kaller scatter()-funksjonen for å sette punkter
plt.scatter(x, y)

# kaller plot()-funksjonen for å plotte regresjonsfunksjonen
plt.plot(x, y_verdier, 'r')

'''
alternativ metode
plt.plot(x, y, '.', x, y_verdier, '-')
'''
# viser plottet
plt.show()

# skriver ut regresjonsfunksjonen
print(f'U(x) = {a:.3f} * e^{b:.3f}.x')
```

Program A.35: Programmet finner eksponential regresjon for CO_2 utslipp i verden.

Output:

```
U(x) = 17705.027 * e^0.017.x
```



Figur A.13: Figuren viser eksponential regresjon for utslipp av CO_2 i verden.

Grafen A.13 viser at eksponential funksjonen ikke tilpasser dataene nøyaktig.

Lineær regresjon

Algoritme

1 Importere biblioteket *Numpy*, og modulen *matplotlib.pyplot*.

2 Definere x- og y-verdier som *NumPy-array*er.

3 Bruke *polyfit()*-funksjonen fra *NumPy*-biblioteket for å finne koeffisienter til polynomisk regresjon. Funksjonen tar følgende parametere (funksjonen tar flere parametere, men vi går bare gjennom de som blir brukt i programmet):

- **xdata:** x-verdier i datasettet
- **ydata:** y-verdier i datasettet
- **deg:** Graden til funksjonen

Vi velger grad til polynomet lik 1 for å få lineær regresjon. Rekkefølgen for koeffisientene er fra koeffisienter med lavest grad til koeffisienter med høyest grad.

4 Bruke *poly1d()*-funksjonen fra *NumPy*-biblioteket for å lage en polynomisk funksjon. Funksjonen tar koeffisientene som parametere og returnerer en funksjon av grad n. Avhengig av antall koeffisienter, får vi andregradsfunksjon, tredjegradsfunksjon, osv.

5 Definere en liste av x-verdier ved å kalle *linspace()*-funksjonen. Disse verdiene skal vi bruke som input-verdi til regresjonslikningen for å plotte den.

6 Sette tittel for plottet ved å kalle *title()*-funksjon fra *matplotlib.pyplot*-modulen.

7 Sette label for x- og y-aksen ved å kalle henholdsvis *xlabel()*- og *ylabel()*-funksjon fra *matplotlib.pyplot*-modulen.

8 Kalle *scatter()*-funksjonen for å sette dataene i koordinatsystemet. *scatter()*-funksjonen tar x- og y-verdiene (koordinat) som parametere, og setter data som punkter i plottet.

9 kalle *plot()*-funksjonen for å plotte regresjonslinjen. Vi velger også farge til linjen ved å legge til 'r' som parameter i funksjonen. 'r' står for rødfarge.

10 Benytte *show()*-funksjonen for å vise plottet.

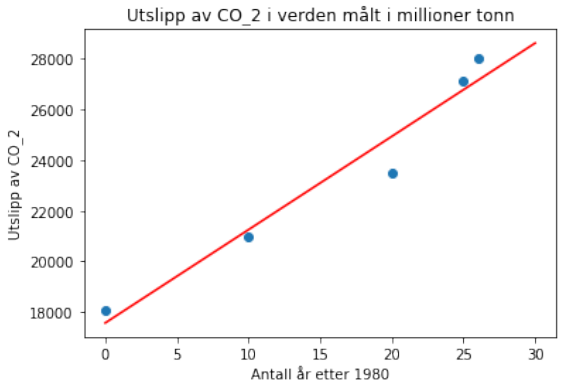
11 Skrive ut funksjonen til regresjonslinjen.

```
'''
program for å finne lineær regresjon for CO_2 utslipp i verden
'''

# importerer biblioteker
import numpy as np
import matplotlib.pyplot as plt

# definerer x-verdier
x = np.array([0, 10, 20, 25, 26])

# definerer y-verdier
y = np.array([18054, 20988, 23509, 27146, 28003])

# polyfit()-funksjonen finner koeffisienter til polynomisk regresjon med grad=1 -> lineær regresjon
koeff = np.polyfit(x, y, 1)

# poly1d()-funksjonen tar koeffisienter og returnerer en n-te gradsfunksjon
U = np.poly1d(koeff)

# lager x-verdier for regresjonsfunksjonen
x_verdier = np.linspace(0, 30, 100)

# setter tittel, x- og y-label
plt.title('Utslipp av CO_2 i verden målt i millioner tonn')
plt.xlabel('Antall år etter 1980')
plt.ylabel('Utslipp av CO_2')

# kaller scatter()-funksjonen for å sette punkter
plt.scatter(x, y)

# kaller plot()-funksjonen for å plotte regresjonsfunksjonen
plt.plot(x_verdier, U(x_verdier), 'r')

# viser plottet
plt.show()

# skrive ut regresjonsfunksjonen
print('U(x)=\n', U)
```

Program A.36: Programmet finner lineær regresjon for CO_2 utslipp i verden.

Output:

```
U(x)=

368.3 x + 1.757e+04
```



Figur A.14: Figuren viser lineær regresjon for utslipp av CO_2 i verden.

Slik vi ser fra grafen A.14, tilpasser ikke lineær funksjonen dataene.

Polynomisk regresjon

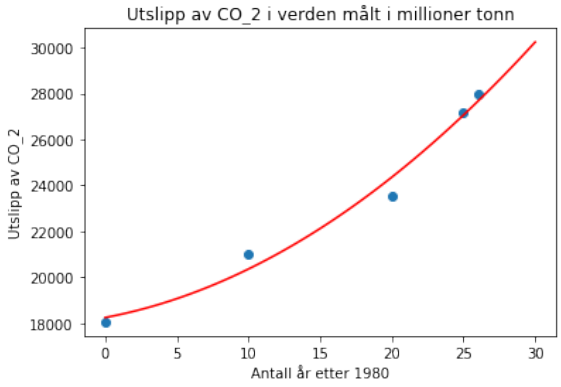
Algoritme Vi bruker samme programkode som vi brukte for lineær regresjon for polynomisk regresjon med forskjell at vi endrer graden til polynomet:

```
koeff = np.polyfit(x, y, n=1, 2, 3, ...)
```

Resten av programmet er den samme som for lineær regresjon. Derfor legger vi ikke til koden for de andre polynomiske regresjoner for å unngå gjentakelse, men Det legges til output for å vise resultater ved modelleringen.

Output av programmet for polynomisk regresjon av grad 2:

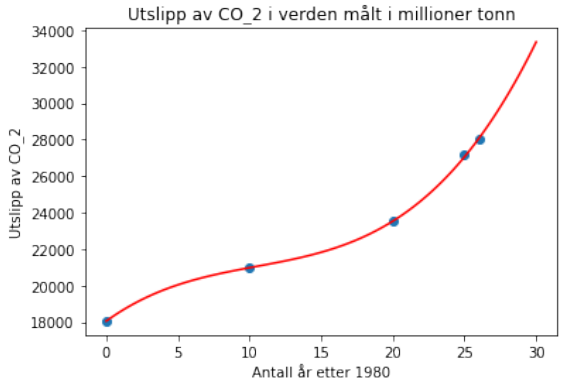
```
U(x)=
        2
9.419 x + 117.4 x + 1.825e+04
```



Figur A.15: Figuren viser polynomisk regresjon av grad 2 for utslipp av CO_2 i verden.

Output av programmet for polynomisk regresjon av grad 3:

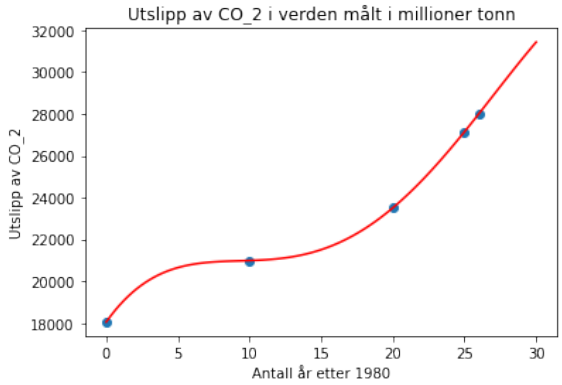
```
U(x)=
         3         2
1.273 x - 40.05 x + 565.8 x + 1.806e+04
```



Figur A.16: Figuren viser polynomisk regresjon av grad 3 for utslipp av CO_2 i verden.

Output av programmet for polynomisk regresjon av grad 4:

```
U(x)=
           4         3          2
-0.07617 x + 5.54 x - 114.9 x + 964.9 x + 1.805e+04
```



Figur A.17: Figuren viser polynomisk regresjon av grad 4 for utslipp av CO_2 i verden.

Vi ser fra figurene A.15, A.16, og A.17 desto økes graden til polynomfunksjonen desto bedre tilpasser regresjonslinjen punkt-dataene. Vi kan sjekke dette ved å sette forskjellige x-verdier i funksjoner vi fikk ved modelleringen. Etter at vi sjekker dette ser vi at resultatet blir mer og mer nøyaktig mens graden til regresjonsfunksjonen øker. For eksempel gir fjerdegrads funksjonen veldig nære resultater som de opprinnelige tallene:

```
[18054.00000000001, 20988.000000000007, 23509.00000000004,
    27146.00000000008, 28003.000000000073]
```

Noe som vi ser også fra plottet for polynomisk regresjon av grad 4. Vi ser at linjen går gjennom alle punktene, dvs. punkt-dataene ligger (nesten) i funksjonens verdimengde.

Utfra dataene som er gitt i oppgaven, og det som vi ser fra grafene, har utslippet av CO_2 hatt en økende tendens. Det vil si det ser ikke ut at landene vil senke utslippet av CO_2 i årene fremover.[13]

[13] Det er mulig at denne modellen ikke viser virkeligheten siden dataene ikke dekker årene etter 2006.

7 Matematikk R2

Kompetansemål etter matematikk R2

Mål for opplæringen er at eleven skal kunne

- utforske egenskaper ved ulike rekker og gjøre rede for praktiske anvendelser av egenskaper ved rekker
- utforske rekursive sammenhenger ved å bruke programmering og presentere egne framgangsmåter
- utvikle algoritmer for å beregne integraler numerisk, og bruke programmering til å utføre algoritmene
- gi eksempler på ulike situasjoner som kan modelleres ved å bruke ulike matematiske funksjoner, og modellere og analysere slike situasjoner ved å bruke reelle datasett

Kilde

7.1 Følger og rekker

Ifølge Bueie (2019, s. 88), *for*-løkker i Python har mye i felles med følger i matematikken, og vi med datastrukturen *liste* kan lage tallfølger på en god måte.

```
en_liste = []  # oppretter en liste
for i in range(0,10,2):   # teller fom 0 tom 10 med steglengde 2
    en_liste.append(i)   # legger tallene i lista

print(en_liste)
```

```
    [0, 2, 4, 6, 8]
```

7.1.1 Aritmetiske tallfølger

Oppgave Lag en aritmetisk tallfølge i Python.

Løsning Vi velger å løse oppgaven ved å ta i bruk formelen A.49 for det n-te leddet i en aritmetisk tallfølge. Vi får første leddet a_1, siste leddet a_s og steglengde d fra brukeren, og bruker en *while*-løkke hvor vi beregner det n-te leddet i hver loop, og legger leddet i en liste. Til slutt skriver vi ut listen.

Algoritme

1 Definere en funksjon kalt *aritmetisk_tallfølge()*.

- 1.1 Få startverdi, sluttverdi, steglengde fra brukeren; konvertere input-verdiene til heltall.
- 1.2 Definere en tom liste for lagring av tallfølgen.
- 1.3 Definere en variabel n som representerer indeksen til et ledd i listen; vi setter variabelen lik 1 i starten.
- 1.4 Definere en variabel a_n for ledd nummer n; vi setter variabelen lik 0 i starten.
- 1.5 Bruke en *while*-løkke som looper så lenge a_n er mindre enn det siste leddet.
  - 1.5.1 Bruke formelen A.49 for å finne n-te ledd i følgen.
  - 1.5.2 Inkrementere indeksen n med 1.
  - 1.5.3 Legge a_n inn i listen ved å bruke *append()*-funksjonen.
- 1.6 Skrive ut listen.

2 Kalle funksjonen *aritmetisk_tallfølge()*.

```
'''
Programmet lager en aritmetisk følge gitt ved intervall og steglengde
'''

def aritmetisk_tallfølge():
    a_1 = int(input('Skriv inn startverdi av tallfølgen: '))   # startverdi i følgen
    a_s = int(input('Skriv inn sluttverdi av tallfølgen: '))    #sluttverdi i følgen
    d = int(input('Skriv inn steglengde: '))    # steglengde

    en_liste = []  # oppretter en liste
    n = 1  # indeks til ledd
    a_n = 0   # n-te ledd i følgen

    while(a_n < a_s): # looper så lenge a_n er mindre enn det siste leddet
        a_n = a_1 + (n - 1) * d  # beregner n-te ledd i følgen
        n += 1 # inkrementerer indeksen med 1
        en_liste.append(a_n)   # legger tallet i lista

    print(en_liste) # skriver ut listen

aritmetisk_tallfølge()
```

Program A.37: Programmet lager en aritmetisk tallfølge gitt ved intervall og steglengde.

Eksempel output:

```
Skriv inn startverdi av tallfølgen: 1
Skriv inn sluttverdi av tallfølgen: 10
Skriv inn steglengde: 3
[1, 4, 7, 10]
```

7.1.2 Aritmetiske rekker

Oppgave I følgende rekke øker hvert ledd med 4:

1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 ...

Lag et program som finner summen av de 100 første tallene i rekka.

Oppgaven er hentet fra (ProFag, 2021).

Løsning Vi har en aritmetisk rekke der første leddet a_1, steglengde d og antall ledd n er kjent. For å finne summen av de 100 første leddene, må vi først finne a_{100} , og så benytte formelen A.52 for summen av de n første leddene i en aritmetisk rekke.

For å finne leddet a_{100}, bruker vi formelen A.49. Vi lager derfor en funksjon som finner ledd a_n i en aritmetisk tallfølge, og en annen for å finne summen av de n første leddene i en aritmetisk rekke.

Algoritme

1 Definere en funksjon *finn_nte_ledd()* som finner n-te ledd i en aritmetisk tallfølge. Funksjonen tar første ledd a_1, steglengde d, ledd-indeks n som parametere.

1.1 Beregne a_n ved å bruke formelen A.49.

1.2 Returnere a_n.

2 Definere *main()*-funksjon som klientprogram; funksjonen finner summen av de n første leddene.

2.1 Få første ledd a_1, steglengde d, ledd-indeks n som input fra brukeren; konvertere de to første input-verdiene til flyttall, og den siste til heltall.

2.2 Finne a_n ved å kalle *finn_nte_ledd()*-funksjonen.

2.3 Finne summen av rekken ved å bruke formelen A.52.

2.4 Skrive ut resultatet.

3 Kalle *main()*-funksjonen.

```
'''
funksjonen finner n-te ledd i en aritmetisk tallfølge
parametere: første ledd a_1, steglengde d, ledd-indeks n'
return: ledd a_n
'''
def finn_nte_ledd(a_1, d, n):

    # beregner a_n med å bruke formelen for n-te ledd i en aritmetisk følge
    a_n = a_1 + (n - 1) * d

    # returnerer resultatet.
    return a_n

# funksjonen finner summen av n første leddene
def main():

    '''
    får input fra brukeren, og konverterer dem til float
    (metoden forutsetter at brukeren ikke skriver brøk-del tall)
    '''
    a_1 = float(input('Oppgi første leddet i rekken: '))
    d = float(input('Oppgi steglengde: '))
    n = int(input('Oppgi antall ledd: '))


    # finner a_n med å kalle på funksjonen finn_nte_ledd
    a_n = finn_nte_ledd(a_1, d, n)

    # finner summen med å bruke formelen for summen av aritmetiske rekker
    S_n = ((a_1 + a_n) / 2) * n

    # Skriver ut resultatet
    print(f'Summen av {n} første leddene i en aritmetisk rekke er {S_n:.2f}')

main()
```

Program A.38: Programmet finner summen av de n første leddene i en aritmetisk rekke.

Output:

```
Oppgi første leddet i rekken: 1
Oppgi steglengde: 4
Oppgi antall ledd: 100
Summen av 100 første leddene i en aritmetisk rekke er 19900.00
```

Alternativ løsning lånet fra ProFag (2021): Programmet bruker en *for*-løkke som looper 99 ganger; i hver loop brukes den rekursive formelen for n-te ledd i en aritmetisk tallfølge (Se A.48), samtidig som leddene summeres.

```
tall = 1    #definerer tall som 1 fordi rekken starter på 1
sum = 1     #definerer sum som 1 fordi første tall er gitt

for i in range(99):     #kjører 99 ganger
    tall = tall + 4           #legger til 4 på forrige tall-verdi
    sum = sum + tall         #legger til tall-verdi på forrige sum-verdi

print(sum)
```

Output:

```
19900
```

7.1.3 Geometriske rekker

Oppgave Lag et program som viser at summen av denne rekka er 4.5

$$3 + 1 + \frac{1}{3} + \frac{1}{9} + \frac{1}{27}$$

Oppgaven er hentet fra (Haraldsrud mfl., 2020).

Løsning Vi ser fra rekken at den er en geometrisk rekke, hvor $a_1 = 3$, $k = \frac{1}{3}$ og $n = 5$. Vi benytter formelen A.53 for å finne ut om summen av rekken er 4.5.

Vi lager en funksjon som tar a_1, k, og n som parametere, og returnerer summen av rekken. I programmet må vi sørge for at k-verdien er forskjellig fra 1, i tilfellet $k = 1$, bruker vi formelen A.54. Dette fikser vi ved å bruke en *if-else*-setning.

Algoritme

1 Definere en funksjon *finn_sum()*, funksjonen tar første ledd a_1, kvotient k, og antall ledd n som parametere.

- 1.1 Definere en variabel S_n, og sette den lik 0.
- 1.2 Hvis $k = 1$
  - 1.2.1 benytte formelen A.54 for å beregne summen.
- 1.3 Ellers:
  - 1.3.1 benytte formelen A.53.
- 1.4 Returnere resultatet.

2 Definere en funksjon *main()* som klientprogram.

- 2.1 Be brukeren om å oppgi første ledd a_1 i rekken, og konvertere den til flyttall.

2.2 Be brukeren om å oppgi kvotient k, her må vi sørge for situasjoner brukeren skriver kvotient som brøk. Vi benytter *try-except* for å håndtere *valueError*.

2.3 Prøve å

2.3.1 konvertere kvotient-inputen til flyttall.

2.4 Dersom det vil skje *valueError*

2.4.1 splitte strengen med '/', og hente ut telleren og nevneren som separate strenger.

2.4.2 konvertere dem til flyttall, og regne ut kvotient-verdien ved å dele telleren på nevneren.

2.5 Be brukeren om å oppgi antall ledd n, og konvertere den til heltall.

2.6 Beregne summen ved å kalle *finn_sum()*-funksjonen.

2.7 Skrive ut resultatet

3 Kalle *main()*-funksjonen.

```python
'''
funksjonen finner summen av en geometrisk rekke
parametere: første ledd a_1, kvotient k, antall ledd n
return: summen av rekka
'''
def finn_sum(a_1, k, n):

    # definerer en variabel S_n for å lagre summen i
    S_n = 0

    # sjekker om k=1
    if k == 1:
        # bruker formelen for summen av geometriske rekker når k=1
        S_n = n * a_1

    else:
        # bruker formelen for summen av geometriske rekker når k er forskjellig fra 1
        S_n = a_1 * ((k**n - 1) / (k -1))

    # returnerer resultatet
    return S_n


# funksjonen kaller finn_sum() og løser oppgaven
def main():

    # får a_1 fra brukeren, konverterer input-verdiene til float
    a_1 = float(input('Skriv inn første leddet i rekken: '))

    # får k fra brukeren
    k = input('Skriv inn kvotienten: ')

    # bruker try-except for å sørge for situasjoner input-verdien er et brøk
    try:
        # hvis input-verdien er et heltall/desimaltall
        k = float(k)

    # håndtering av valueError
    except ValueError:

        # splitter kvotient-input delene
        teller, nevner = k.split('/')

        # konverterer hver del til float og deler teller til nevner
        k = float(teller) / float(nevner)

    # får n fra brukeren, konverterer til int
    n = int(input('Skriv inn antall ledd: '))

    # kaller funksjonen finn_sum() og beregner summen
    S_n = finn_sum(a_1, k, n)

    # skriver ut resultatet
    print(f'Summen av de {n} første leddene i rekken er {S_n:.1f}')

main()
```

Program A.39: Programmet finner summen av en de n første leddene i en geometrisk rekke.

Output:

```
Skriv inn første leddet i rekken: 3
Skriv inn kvotienten: 1/3
Skriv inn antall ledd: 5
Summen av de 5 første leddene i rekken er 4.5
```

7.2 Integrasjon

Begrepet integral refereres som regel til summering av uendelige stykker for å finne innholdet i et kontinuerlig område. Prosessen med å beregne en integral kalles integrasjon, og den tilnærmede beregningen av en integral kalles numerisk integrasjon (Weisstein, 2002a).

Oppgave Finn en numerisk tilnærming for det bestemte integralet ved hjelp av Rektangelmetoden.

$$\int_2^5 \sqrt{x}\, dx \tag{A.21}$$

Løsning Når vi skal løse bestemte integraler numerisk, kan vi regne ut arealet under en funksjonsgraf med utgangspunkt i en funksjon.

Fremgangsmåten for rektangelmetoden er å dele arealet under grafen i flere rektangler, og finne summen av disse rektanglene; desto flere rektangler setter vi inn, desto mer nøyaktig svar får vi. Vi benytter formelen for rektangelmetoden A.47 for å løse oppgaven.

Vi har allerede verdien til a og b, det som ikke er oppgitt, er antall rektangler n. Denne verdien skal vi få fra brukeren, deretter kan vi regne ut bredden h til rektangler. Lengden for hvert rektangel er lik $f(x_i)$ der $i = 1, 2, 3, \quad ..., n$. Summen av arealet til alle de n rektanglene er det bestemte integralet av $f(x) = \sqrt{x}$ fra $a = 2$ til $b = 5$. Se figur A.18 som illustrerer bruk av rektangelmetoden med 10 rektangler for å løse integralet.

Alternativt kan vi løse oppgaven ved å bruke *sum()*-funksjonen fra *NnmPy*-biblioteket. Vi løser oppgaven med begge metodene, også legger vi til et plott som viser rektangelmetoden grafisk. [14]

Algoritme

1 Importere biblioteket *Numpy*, og *matplotlib.pyplot*-modulen.

2 Definere en funksjon $f()$ som returnerer $\sqrt{x}$.

3 Definere en funksjon *rektangelmetoden()*.

- 3.1 Få antall rektangler n fra brukeren; konvertere input-verdien til heltall.
- 3.2 Definere variablene a og b med de gitte verdiene ($a = 2$, $b = 5$).
- 3.3 Definere en variabel h som bredde for hvert rektangel. For dette bruker vi formelen A.46.
- 3.4 Definere en variabel *total_sum* for å lagre resultatet i.
- 3.5 Bruke en *for*-løkke som tar *range()*-funksjonen med n som parameter.
  - 3.5.1 Definere x lik $a + i * h$ der a er startpunktet av intervallet, og i indeksen for x.
  - 3.5.2 Beregne areal for hvert rektangel gitt ved lengden $f(x)$ $\times$ bredden h. Vi summerer arealene fortløpende mens vi looper gjennom løkken.
- 3.6 Skrive ut resultatet.
- 3.7 Definere og lage en liste for x-verdier ved å bruke *linspace()*-funksjonen. Funksjonen tar a, $b - h$ og n som parametere. (b er ikke inkludert i listen siden vi beregner venstre Riemann-sum).
- 3.8 Kalle funksjonen *sum()* fra *NumPy*; funksjonen returnerer summen av arealene til alle rektanglene. I vårt tilfelle er arealet lik $f(x\_verdier) * h$.

[14] Løsningsforslaget for denne delen er hentet fra (Walls, 2019b).

3.9 Skrive ut resultatet.

3.10 Definere og lage en liste for y-verdier; vi kaller *f()*-funksjonen og setter x-verdier som parameter for funksjonen.

3.11 Plotte grafen til $f(x)$ ved hjelp av *plot()*-funksjonen. Funksjonen tar x- og y-verdier som parametere. Det er også mulig å sette farge til grafen ved å skrive farge som streng som parameter.

3.12 Lage liste over venstre x- og y-verdier; for eksempel lager vi listen for venstre x-verdier med følgende syntaks:

```
x_venstre = x_verdier[:-1]
```

3.13 Sette tittel for plottet ved å kalle *title()*-funksjonen fra *pyplot*.

3.14 Sette label for x- og y-aksen ved å kalle henholdsvis *xlabel()*- og *ylabel()*-funksjonen.

3.15 Kalle *bar()*-funksjonen fra *Pyplot* for å plotte rektangler; funksjonen tar følgende parametere (Hunter mfl., 2021e):

- **x:** x-verdier
- **height:** y-verdier
- **width:** h, bredde for rektangler
- **alpha:** bestemmer gjennomsiktighets graden
- **align:** justerer stolpene, standard er 'center', med 'edge' tar vi venstre kant på stolpene
- **edgecolor:** farge til kantene

3.16 Vise plottet ved å kalle *show()*-funksjonen.

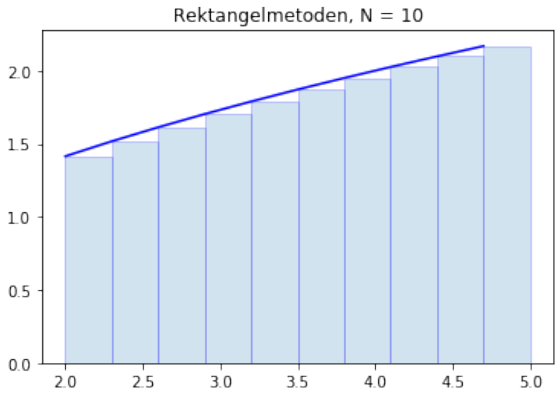
4 Kalle *rektangelmetoden()*.

```python
'''
Programmet implementerer Rektangelmetoden (venstre Riemann-sum)
og løser funksjonen f(x) vha. metoden
'''

# importerer biblioteker
import matplotlib.pyplot as plt
import numpy as np

# funksjon som skal integreres
def f(x):
    return np.sqrt(x)

# definerer funksjonen for rektangelmetoden
def rektangelmetoden():
    n = int(input('Skriv inn antall rektangler: ')) # får verdien for antall rektangler

    # setter opp innstillinger
    a = 2                       # startverdi
    b = 5                       # sluttverdi
    h = (b - a)/n               # bredde
    total_sum = 0               # variabel for å lagre total summen

    for i in range(n): # vi looper gjennom løkken fra 0 til n
        x = a + i * h  # verdien av x_i er lik startverdi + i * bredde

        # areal av en rektangel er lik lengde * bredde her f(x_i) = lengde for i-te rektangelet
        areal = f(x) * h

        # summerer areal for hver rektangel inn i variabelen total_sum
        total_sum += areal

    # skriver ut resultatet
    print(f'Numerisk verdi av integralet (beregnet av egendefinert funksjon) er lik {total_sum:.3f}')

    '''
    Alternativ metode samt tegning av grafen:

    Vi lager et sett av x- og y-verdier, disse verdiene brukes for både tegning av grafen og beregning av summen
    x_verdier lager vi ved å kalle linspace()-funksjonen;
    b er ikke inkludert i x-verdier siden vi beregner venstre Riemann-sum
    '''
    x_verdier = np.linspace(a,b-h,n)
    '''
    kaller funksjonen sum() fra NumPy, funksjonen returnerer summen av arealene til alle rektangler
    her  f(x_verdier) * h = arealet til rektangler
    '''
    venstre_riemann_sum = np.sum(f(x_verdier) * h)

    # skriver ut summen
    print(f'Numerisk verdi av integralet (beregnet av sum()-funksjonen fra NumPy) er lik {total_sum:.3f}')

    y_verdier = f(x_verdier)  # lager en liste av y-verdier
    plt.plot(x_verdier,y_verdier, 'blue')  # tegner grafen til f

    # lager liste over venstre x- og y-verdier
    x_venstre = x_verdier[:-1]
    y_venstre = y_verdier[:-1]

    # setter tittel for grafen, format()-funksjonen benyttes for å skrive ut n-verdien på grafen
    plt.title('Rektangelmetoden, N = {}'.format(n))

    # setter label for x- og y-aksen
    plt.xlabel('x')
    plt.ylabel('y')
    '''
    bar-funksjonen tar følgende parametere:
    matplotlib.pyplot.bar(x, height, width=0.8, bottom=None, *, align='center', data=None, **kwargs)[source]
    x = x-verdier
    height = y_verdier
    width = h
    alpha = bestemmer gjennomsiktighetsgraden
    align = justerer stolpene, standard er 'center'. med 'edge' tar vi venstre kant på stolpene etter x-verdier
    edgecolor = farge til kantene
    '''
    plt.bar(x_verdier,y_verdier, h, alpha=0.2, align='edge',edgecolor='blue')
    plt.show()  # viser plottet

rektangelmetoden()
```

Program A.40: Programmet implementerer Rektangelmetoden (venstre Riemann-sum).

Output:

```
Skriv inn antall rektangler: 10
Numerisk verdi av integralet (beregnet av egendefinert funksjon) er lik 5.444
Numerisk verdi av integralet (beregnet av sum()-funksjonen fra NumPy) er lik 5.444
```



Figur A.18: Illustrasjon for bruk av rektangelmetoden for å løse integralet A.21.

7.3 Differensiallikninger

"En differensialligning er en ligning som relaterer derivatene til en ukjent funksjon, selve funksjonen, variablene som funksjonen defineres med, og konstanter. Hvis den ukjente funksjonen avhenger av en enkelt reell variabel, kalles differensiallikningen en ordinær differensialligning"(Farlow, 2006, s. 10). En av de mest velkjente differensiallikninger er Bernoulli likning:

$$\frac{dy}{dx} + y = y^2 \tag{A.22}$$

Her kalles den ukjente størrelsen $y = y(x)$ den avhengige variabelen, og den reelle variabelen x, kalles den uavhengige variabelen (Farlow, 2006).

Løsninger av differensiallikninger kan sees som modeller som beskriver forskjellige fenomener som en funksjon (ofte som en funksjon av tid). "Å sette opp og løse differensiallikninger kalles derfor også for å *modellere*"(Kristensen & Aanensen, 2018b).

Differensiallikninger er brukelig overalt innen vitenskap og samfunnsliv hvor vi har behov for å finne modeller som beskriver sammenheng mellom størrelser. Derfor regnes differensiallikninger som et av viktigste områdene innenfor matematikkfaget (Kristensen & Aanensen, 2018b).

"En løsning av en n-ordens differensialligning er en n ganger differensierbar funksjon $y = y(x)$, som, når den substitueres i ligningen, tilfredsstiller den identisk over et intervall $a < x < b$, vil vi si at funksjonen y er en løsning av differensiallikningen over intervallet $a < x < b$"(Farlow, 2006, s. 15).

7.3.1 Første ordens differensiallikninger

En første ordens differensiallikning har generell form som:

$$\frac{dy}{dt} = f(y, t) \tag{A.23}$$

hvor dy/dt er endringer i y med hensyn til tid t, og $f(y,t)$ er en hvilken som helst funksjon av y og t.

Vi tar for oss en av de enkleste former for differensiallikninger for å vise hvordan vi kan løse en første ordens differensiallikning analytisk:

$$\frac{dy}{dt} = -y$$

Vi begynner med å omorganisere ligningen:

$$\frac{dy}{y} = -dt$$

Med å løse en differensiallikning, mener vi å finne et uttrykk for $y(t)$. Vi integrerer begge sider og får:

$$ln(y) = -t + C$$

tar eksponential av begge sider for å finne et uttrykk for y:

$$e^{ln(y)} = e^{-t+C}$$

og får:

$$y = Ce^{-t}$$

Hvis vi har en initialverdi for y, kan vi finne en verdi for C også (B. D. Storey, udatert).

7.3.2 Andre ordens differensiallikninger Det enkleste eksempelet av et andre ordens differensiallikning er et lodd med masse m som henger i en fjær hvor tyngdekraften mg er normal til bevegelses retningen.

Hvis vi trekker loddet nedover og slipper det, vil loddet svinge opp og ned om likevektsstillingen. Ifølge Newtons andre lov er likningen for systemet gitt ved:

$$F = ma \tag{A.24}$$

hvor F er kraften (fjærkraften) som virker på masse m og a er akselerasjon. Fjærkraften, kraften på loddet fra fjæren, er proporsjonal med lengden og er gitt ved:

$$F = -kx \tag{A.25}$$

Her k er fjærkonstanten og x er forskyvningen til fjæren fra likevektsstillingen. Vi erstatter F med formelen for fjærkraften og får:

$$ma = -kx \tag{A.26}$$

Vi vet at akselerasjon er den andre deriverte av posisjon $\frac{d^2x}{dt^2}$, så vi vil ha en andre ordens

differensiallikning som følger:

$$m\frac{d^2x}{dt^2} = -kx \iff \frac{d^2x}{dt^2} = \frac{-k}{m}x \tag{A.27}$$

Vi kan løse differensiallikningen analytisk ved å finne røtter til den karakteristiske likningen. Avhengig av hvor mange røtter vi får, vil den generelle løsningen for differensiallikningen være annerledes.

7.3.3 Built-in funksjoner i Python I Python kan vi løse differensiallikninger numerisk ved hjelp av *ODEINT()*-funksjonen fra *scipy.integrate*-modulen.

ODEINT()-funksjonen tar flere parametere, men her tar vi de tre første parameterne som brukes oftest for å løse en differensiallikning:

```
scipy.integrate.odeint(func, y0, t)
```

- **func:** Funksjonen som returnerer verdier basert på y ved tiden t.
- **y0:** Initialtilstand for y, kan også være en vektor av initialverdier.
- **t:** En rekke av tidspunkter som skal løses for y. Startpunktet bør være det første elementet i listen. Sekvensen må være monotont økende eller monotont synkende; gjentatte verdier er tillatt (The-SciPy-community, 2021a).

Oppgave En vannbeholder inneholder 6000 liter væske. Vi åpner en kran slik at 30 liter væske renner ut per minutt. La y være den mengde væske som til enhver tid er igjen på tanken. Vi åpner kranen ved $t = 0$. Væsken renner ut med en konstant mengde på 30 liter per minutt, dvs.:

$$\frac{dy}{dt} = -30$$

Ved $t = 0$ var væskemengden 6000 L. Løs differensiallikningen, og beregn når tanken går tom.

Oppgaven er hentet fra (Kristensen & Aanensen, 2018b).

Løsning Vi bruker *ODEINT()*-funksjonen for å løse differensiallikningen, og så finner vi likningen $y(t)$ og setter den lik 0 for å finne ut når tanken går tom. Vi kan også plotte funksjonen og se fra grafen når tanken går tom.

Algoritme

1 Importere biblioteket *Numpy*, modulen *matplotlib.pyplot*, og funksjoner *odeint()* og *solve()* fra henholdsvis *scipy.integrate*-modulen og *SymPy*-biblioteket.

2 Definere en funksjon *f()*; funksjonen tar y og t som parametere.

  2.1 Returnerer en funksjon på formen $dy/dt = f(y, t)$.

3 Definere initialverdien for y ved tiden $t = 0$.

4 Definere en liste av tidspunkter (t-verdier); vi velger et intervall fra 0 til 200, med 10 punkter. Disse verdiene kan variere.

5 Løse differensiallikningen ved hjelp av *odeint()*-funksjonen; funksjonen tar funksjonen $f()$, initialverdi y_0, og tidsverdiene som parametere, og returnerer en liste av y-verdier.

6 Kalle funksjonen *polyfit()* fra *NumPy*-biblioteket for å finne koeffisientene til likningen $y(t)$. Vi velger her en polynomfunksjon av grad 1.

7 Bruke *poly1d()*-funksjonen fra *NumPy*-biblioteket for å få likningen $y(t)$. Funksjonen tar koeffisienter og variabel (til funksjonen) som parametere, og returnerer en n-te gradslikning.

8 Skrive ut likningen $y(t)$

9 Finne ut når tanken går tom, ved å benytte *solve()*-funksjonen fra *SymPy*-biblioteket. Funksjonen tar likning og variabel som parametere, og returnerer resultatet; skrive ut resultatet.

10 Plotte funksjonen ved å bruke *plot()*-funksjonen. Funksjonen tar t-verdier og y-verdier som parametere.

11 Sette label for x- og y-aksen ved å kalle henholdsvis *xlabel*- og *ylabel*-funksjonen.

12 Vise plottet ved å kalle *show()*-funksjonen.

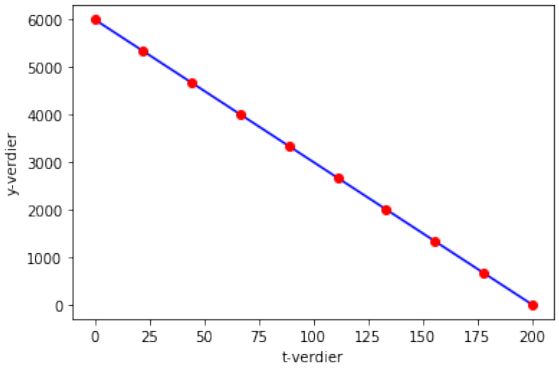
```
'''
Programmet løser første ordens differensiallikningen f(y, t)
'''

# importerer biblioteker
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import numpy as np
from sympy import solve

# definerer funksjonen dy/dt=f(y, t), og returnerer den
def f(y, t):
    dydt = -30
    return dydt

# definerer initialverdi for y -> y(0)
y0 = 6000

# definerer t-verdier fra 0 til 200, antall punkter 10
t_verdier = np.linspace(0, 200, 10)

'''
løser differensiallikningen ved å kalle på odeint()-funksjonen
parametere: funksjonen f(y, t), initialverdi y0, 10 input-verdier(t-verdier)
returnerer y-verdier (for de 10 tidspunktene vi passerte som parameter)
'''
y_t = odeint(f, y0, t_verdier)

'''
finner koeffisienter til en funksjon som passer y-verdier (velger polynom av grad 1)
koeffisientene er en nx1 matrise avhengig av polynomgraden
'''
koeff = np.polyfit(t_verdier, y_t, 1)

'''
finner funksjonen y(t) ved å kalle på poly1d()-funksjonen
parametere: koeffisienter, variabel
returnerer funksjonen y(t)
'''
y = np.poly1d(koeff[:,0], variable='t')

# skriver ut funksjonen
print('y(t)=\n', y)

# setter y(t)=0, og finner t
print('\ny(t) = 0 -> t = ', solve(-30*t + 6000, t))

# første linjen plotter funksjonen i blå, andre linje plotter punkter for y-verdier i rød
plt.plot(t_verdier, y_t, '-b')
plt.plot(t_verdier, y_t, 'ro')

# setter label for x- og y-aksen
plt.xlabel('t-verdier')
plt.ylabel('y-verdier')

# viser plottet
plt.show()
```

Program A.41: Programmet løser første ordens differensiallikningen $f(y, t)$.

Output:

```
y(t)=

-30 t + 6000
```

```
y(t) = 0 -> t =  [200]
```



Figur A.19: Grafen viser at tanken er tom etter 200 minutter.

Vi ser både fra output av programmet og grafen A.19 at vannbeholderen vil gå tom etter 200 minutter.

For å løse andre ordens differensiallikninger med *odeint()*-funksjonen, må vi først konvertere den til et system med første ordens differensialligninger.

Oppgave En vannbeholder inneholder 6000 liter væske. La $y(t)$ være den mengde væske som til enhver tid t (i timer) er igjen på tanken. Vi åpner kran slik at differensiallikningen som modellerer det fenomenet er gitt ved:

$$\frac{d^2y(t)}{dt^2} = -10, \quad y(0) = 6000, \quad y'(0) = 0$$

Løs differensialligningen, og beregn når tanken går tom.

Oppgaven er hentet fra (≪MAT20x Vår 2021≫, 2021).

Løsning Vi har
$y''(t) = -10$
setter
$y'(t) = u(t)$
$u'(t) = -10$

da har vi laget et system av likninger. Nå kan vi løse dette systemet i Python.

Algoritme

1 Importere biblioteket *Numpy*, modulen *matplotlib.pyplot*, og funksjoner *odeint()* og *Table()* fra henholdsvis *scipy.integrate*- og *astropy.table*-modulen.

2 Definere funksjonen $f(u, t)$ der u er en to-dimensjonal liste (en liste av lister) av likninger; funksjonen tar altså u og t som parametere.

2.1 Returnere en funksjon på formen $dy/dt = f(u, t)$.

3 Definere initialverdier som en liste der den første verdien er initialverdien for y, og den andre verdien er initialverdien for y'.

4 Definere tidsverdier fra 0 til 35 med 700 punkter ved å kalle *linspace()*-funksjonen; disse verdiene kan variere.

5 Løse differensiallikningen ved å kalle *odeint()*-funksjonen, funksjonen tar funksjonen $f(u, t)$, initialverdi y_0, og t-verdier som parametere, og returnerer u-verdier (en liste av lister hvor hver av de listene har $y(t)$ og $y'(t)$ som elementer).

6 Definere en variabel for $y(t)$-verdier, og sette den lik første verdien i hver liste i u_t-listen.

7 Plotte y-verdier mot t-verdier ved hjelp av *plot()*-funksjonen.

8 Sette label for x- og y-aksen ved å kalle henholdsvis *xlabel()*- og *ylabel()*-funksjonen.

9 Vise plottet ved å kalle *show()*-funksjonen.

10 Sette y- og t-verdier i en tabell for å finne ut når tanken går tom. Vi benytter funksjonen *Table()* fra *astropy.table*-modulen. Funksjonen tar tidsverdier, y-verdier og navn for hver kolonne som parametere, og returnerer tabellen; Vi avrunder verdier til to desimaler.

11 Skrive ut tabellen.

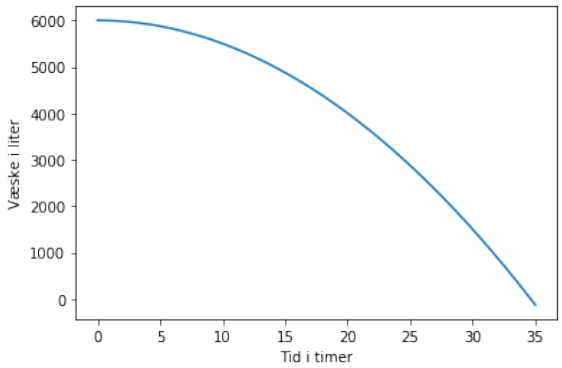
```
'''
Programmet løser andre ordens differensiallikningen f(u, t)
'''

# importerer biblioteker
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import numpy as np
from astropy.table import Table

'''
definerer funksjonen f(u, t) der u er en to-dimensjonal liste (en liste av lister)
som består av u(t) og u'(t), vi laget følgende systemet:
y''(t)=-10
y'(t)=u(t)
u'(t)=-10
'''
def f(u, t):
    dydt = (u[1], -10)
    return dydt

# definerer initialverdier som en liste; y(0)=6000, y'(0)=0
y0 = [6000, 0]

# definerer t-verdier fra 0 til 35, antall punkter 700
t_verdier = np.linspace(0, 35, 700)

'''
løser differensiallikningen ved å kalle på odeint()-funksjonen
parametere: funksjonen f(u, t), initialverdi y0, t-verdier
returnerer u-verdier
'''
u_t = odeint(f, y0, t_verdier)

# løsningen y(t) er lik den første verdien fra hver liste
y_t = u_t[:,0]

# plotter y-verdier mot t-verdier
plt.plot(t_verdier, y_t)

# setter label for x- og y-aksen
plt.xlabel('Tid i timer')
plt.ylabel('Væske i liter')

# viser plottet
plt.show()

# setter verdier i en tabell for å finne ut når tanken blir tom
tabell = Table([np.round(t_verdier, decimals=2), np.round(y_t, decimals=2)],
names=('Tid i timer', 'Væske i liter'))

# skriver ut tabellen
print(tabell)
```

Program A.42: Programmet løser andre ordens differensiallikningen $f(u, t)$.

Output:

```
Tid i timer Væske i liter
----------- -------------
        0.0        6000.0
       0.05       5999.99
        0.1       5999.95
       0.15       5999.89
        0.2        5999.8
       0.25       5999.69
        0.3       5999.55
```

```
 0.35      5999.39
  0.4       5999.2
 0.45      5998.98
  ...          ...
 34.5         49.0
34.55        31.71
 34.6         14.4
34.65        -2.94
 34.7        -20.3
34.75       -37.69
 34.8        -55.1
34.85       -72.54
 34.9        -90.0
34.95      -107.49
 35.0       -125.0
```



Figur A.20: Grafen viser at tanken vil gå tom etter omtrent 34.60 timer.

Vi ser både fra tabellen og fra grafen A.20 at tanken vil gå tom etter omtrent 34.60 timer. Vi har altså funnet en tilnærmet løsning for differensiallikningen.

7.3.4 Eulers metode Vi bruker Eulers metode for å finne en tilnærmet løsning for en første ordens differensiallikning. Metoden kan brukes for differensiallikninger av høyere orden også, men vi må først omforme likningen til et system består av første ordens likninger. Generelt kan en differensiallikning av orden n gjøres om til n likninger av orden 1.

Vurder en første ordens differensialligning med en initialtilstand:

$$y' = f(y,t) \ , \quad y(t_0) = y_0 \tag{A.28}$$

Eulers metode tar utgangspunkt i en startverdi y_n, og finner en tilnærmet verdi for y_{n+1}:

$$y_{n+1} = y_n + f(y_n, t_n)(t_{n+1} - t_n) \tag{A.29}$$

Vi bruker denne formelen for å implementere en funksjon *Eulers_metode()* i Python. Funksjonen tar en differensiallikning $f(y,t)$, y_0, og t-verdier som parametere, og returnerer en liste av y-verdier. De verdiene er som sagt tilnærminger til den eksakte løsningen. Programkoden for implementasjon av Eulers metode er lånet fra Walls (2019a).

I delkapittelet om første ordens differensiallikninger så vi på likningen

$$\frac{dy}{dt} = -y$$

Likningen hadde en generell løsning lik

$$y = Ce^{-t}$$

Hvis vi setter en initialverdi $y(0) = 1$, får vi:

$$f(y,t) = e^{-t}$$

La løse differensiallikningen numerisk ved å bruke Eulers metode.

Algoritme for Eulers metode

1 Definere en funksjon *Eulers_metode()*; metoden tar en funksjon $f()$, initialverdi y_0 og t-verdier som parametere.

1.1 Definere en liste (matrise) for å lagre y-verdier, ved å kalle *zeros()*-funksjonen fra *NumPy*-biblioteket. Funksjonen tar et heltall n (eller en *tuple* $[n, n]$) som parameter og returnerer en $n \times n$ matrise. Her setter vi n lik lengden for t-verdier.

1.2 Sette initialverdien y_0 lik startverdi i listen for y-verdier (på indeks 0).

1.3 Bruke en *for*-løkke som looper fra 0 til (lengde til t-verdier)-1; siden vi starter fra 0, trekker vi 1 fra lengden til t-verdi-listen.

1.3.1 Bruke formelen A.29, og finne y_{n+1}.

1.4 Returnere y.

```
'''
Programmet implementerer Eulers metoden (lånet fra github-siden til Patrick Walls)
parametere: funksjon f(y, t), initialverdi y0, tidsverdier
returnerer en liste av y-verdier
'''
def Eulers_metode(f, y0, t):

    '''
    lager en matrise (liste) for y-verdier;
    alle verdiene er lik 0;
    lengden på listen er lik lengden for t-verdier
    '''
    y = np.zeros(len(t))

    # setter initialverdien y0 lik startverdi i listen (på indeks 0)
    y[0] = y0

    # looper fra 0 til (lengde til t-verdier)-1 (siden vi starter fra 0)
    for n in range(0, len(t)-1):

        # finner y_n+1 i hver loop med hjelp av formelen for Eulers metode
        y[n+1] = y[n] + f(y[n], t[n]) * (t[n+1] - t[n])
    return y
```

Program A.43: Programmet implementerer Eulers metode.

Algoritme for eksempel bruk av Eulers metode

1 Importere biblioteket *Numpy*, og modulen *pyplot*.

2 Definere en funksjon $f(y,t)$ som returnerer funksjonen $\frac{dy}{dt} = -y$.

3 Definere en liste for t-verdier ved å kalle *linspace()*-funksjonen fra *Numpy*-biblioteket. Vi velger et tidsintervall fra 0 til 10, med 20 punkter.

4 Definere initialverdi y_0, og sette den lik 1.

5 Beregne tilnærmede y-verdier ved å kalle *Eulers_metode()*-funksjonen. Vi lagrer disse i en variabel kalt *y_approx*.

6 Beregne eksakte y-verdier ved å mate den eksakte løsningen $f(y,t) = e^{-t}$ med t-verdier. Vi lagrer disse i en variabel kalt *y_eksakt*.

7 Sette tittel for plottet ved å kalle *title()*-funksjonen fra *pyplot*.

8 Sette label for x- og y-aksen ved å kalle henholdsvis *xlabel()*- og *ylabel()*-funksjonen.

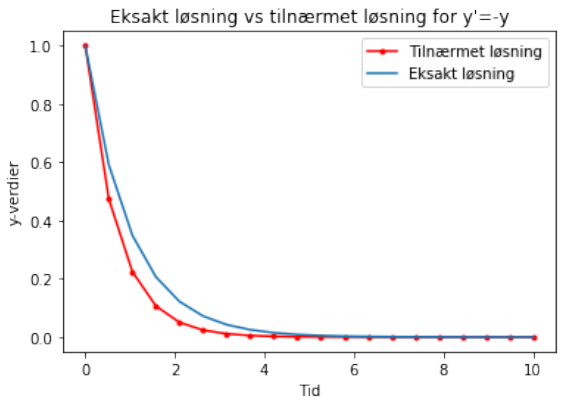
9 Plotte grafen for den eksakte løsningen mot den tilnærmede løsningen, for dette bruker vi *plot()*-funksjonen fra *matplotlib.pyplot*-modulen.

10 Sette label for grafene ved å kalle *legend()*-funksjonen; funksjonen tar en liste av strenger som parameter.

11 Vise plottet ved hjelp av *show()*-funksjonen.

```
'''
Programmet finner tilnærmet løsning for y'=-y vha. Eulers metode
'''

# importerer biblioteker
import numpy as np
import matplotlib.pyplot as plt

# definerer en funksjon f(y, t) som returnerer vår funksjon
def f(y, t):
    dydt = -y
    return dydt

# lager en liste for t-verdier
t_verdier = np.linspace(0, 10, 20)

# initialverdi for y
y0 = 1

# kaller Eulers_metode() for å finne tilnærmede y-verdier
y_approx = Eulers_metode(f, y0, t_verdier)

# lager en liste av eksakte y-verdier
y_eksakt = np.exp(-t_verdier)

# setter tittel, og label for x- og y-aksen
plt.title("Eksakt løsning vs tilnærmet løsning for y'=-y")
plt.xlabel('Tid')
plt.ylabel('y-verdier')

# plotter grafen for den eksakte løsningen mot den tilnærmede løsningen
plt.plot(t_verdier, y_approx, 'r.-', t_verdier, y_eksakt)

# setter label for grafer
plt.legend(['Tilnærmet løsning','Eksakt løsning'])

# viser plottet
plt.show()
```

Program A.44: Programmet finner tilnærmet løsning for $y' = -y$ vha. Eulers metode.



Figur A.21: Figuren illustrerer eksakt løsning vs. tilnærmet løsning for likningen $y' = -y$.

Grafen A.21 viser at Eulers metode har avvik fra den eksakte løsningen på tidspunkter fra 0 til 5, men etter på blir tilnærmingene veldig nære til de eksakte verdier.

Hvis vi velger steglengde for t-verdier så liten som mulig, vil feilen bli mindre.

7.4 Funksjoner og modellering

Oppgave I denne oppgaven skal vi modellere populasjonen til rein. Vi vet at:

- Vi har en reinstamme med 300 dyr.
- Bæreevnen er 200 dyr.
- Naturlig vekst er 10% per år.
- Vekstfarten er gitt ved:

$$y' = 0.10y * (1 - \frac{y}{200}) \tag{A.30}$$

Lag en graf som viser utviklingen til reinpopulasjonen de første 20 årene. Finn ut når tallet på rein er ca. 250, kommenter hva du ser.

Oppgaven er hentet fra (ProFag, 2021).

Løsning Ved tidspunkt $t = 0$ har vi 300 dyr, altså $y_0 = 300$. Oppgaven er å vise utviklingen for de 20 første årene, dvs. vi skal vise utviklingen på tidsintervallet $[0, 20]$. Vi bruker *linspace()*-funksjonen for å lage en liste for tidsverdier, vi setter startpunkt lik 0 og sluttpunkt lik 20 med 20 punkter.

Vi løser oppgaven både ved å bruke *odeint()*-funksjonen og ved å bruke *Eulers_metode()*-funksjonen fra forrige kapittelet om differensiallikninger. Så sammenligner vi svarene vi får fra de to metodene. Vi plotter også grafer til begge metodene.

Algoritme

1 Importere biblioteket *Numpy*, modulen *matplotlib.pyplot*, og funksjoner *solve()*, *odeint()*, og *Table()* fra henholdsvis *sympy*-biblioteket, og *scipy.integrate*- og *astropy.table*-modulen.

2 Definere en funksjon $f(y, t)$ som returnerer funksjonen A.30.

3 Lage en liste for tidsverdier ved å kalle *linspace()*-funksjonen; funksjonen tar startverdi, sluttverdi og antall punkter som parametere.

4 Definere en variabel y_0, og sette den lik 300.

5 kalle *Eulers_metode()* for å finne tilnærmede y-verdier, funksjonen tar f, y_0 og t-verdier som parametere.

6 Kalle *odeint()*-funksjonen for å finne eksakte y-verdier; funksjonen tar f, y_0 og t-verdier som parametere.

7 Finne koeffisienter til funksjonen som passer y-verdier ved å kalle *polyfit()*-funksjonen fra *NumPy*-biblioteket. Funksjonen tar x- og y-verdier, og polynomgraden som parametere. Vi velger polynomgraden lik 2.

8 Finne funksjonen $y(t)$ ved å kalle *poly1d()*-funksjonen; funksjonen tar koeffisientene og variabelen t som parametere, og returnerer funksjonen $y(t)$. Syntaksen er som følger:

```
poly1d(koeff[:,0], variable='t')
```

9 Skrive ut funksjonen $y(t)$.

10 Sette størrelse for plottet ved å kalle *figure()*-funksjonen fra *pyplot*. Vi definerer figurstørrelse som parameter til funksjonen:

```
figure(figsize=(x,y))
```

11 Sette tittel for plottet ved å kalle *title()*-funksjonen fra *pyplot*.

12 Sette label for x- og y-aksen ved å kalle henholdsvis *xlabel()*- og *ylabel()*-funksjonen.

13 Plotte grafen for den eksakte løsningen mot den tilnærmede løsningen ved å kalle *plot()*-funksjonen.

Vi kan sette label for grafen også. Syntaksen kan se ut som følger:

```
plot(x, y,'r.-', label='label')
```

14 Plotte linjen for $y = 250$, denne linjen vil hjelpe oss for å finne når reinpopulasjonen er lik 250. For å plotte horisontal linje benytter vi funksjonen *xhline()* fra *pyplot*-modulen. Vi setter y-verdi, farge og label til grafen som parametere. Det er også mulig å definere intervall for grafen (Hunter mfl., 2021d):

```
axhline(y=0, xmin=0, xmax=1, **kwargs)
```

15 Sette label for grafer ved å kalle *legend()*-funksjonen.

16 Sette rutenett for plottet ved å kalle *grid()*-funksjonen.

17 Vise plottet ved å kalle *show()*-funksjonen.

18 Sette verdier i en tabell for å finne ut når reinpopulasjonen er 250. For dette kaller vi funksjonen *Tabell()*; funksjonen tar flere parametere (Astropy-Developers, 2021):

```
Table(data=None, masked=False, names=None, dtype=None, meta=
    None, copy=True, rows=None, copy_indices=True, units=None,
    descriptions=None, **kwargs)
```

Som data setter vi inn tidsverdier, og eksakte og tilnærmede verdier. På grunn av verdier har flere desimaler, avrunder vi verdiene ved å kalle *round()*-funksjonen fra *Numpy* siden vi har Numpy-arrayer. Vi setter også navn til hver kolonne som parameter. Syntaksen kan se slik ut:

```
Table([np.round(t_verdier, decimals=0), np.round(y_eksakt,
    decimals=0), np.round(y_approx, decimals=0)],names=(' Aar',
    'Eksakt verdi','Tilnaermet verdi'))
```

19 Skrive ut tabellen.

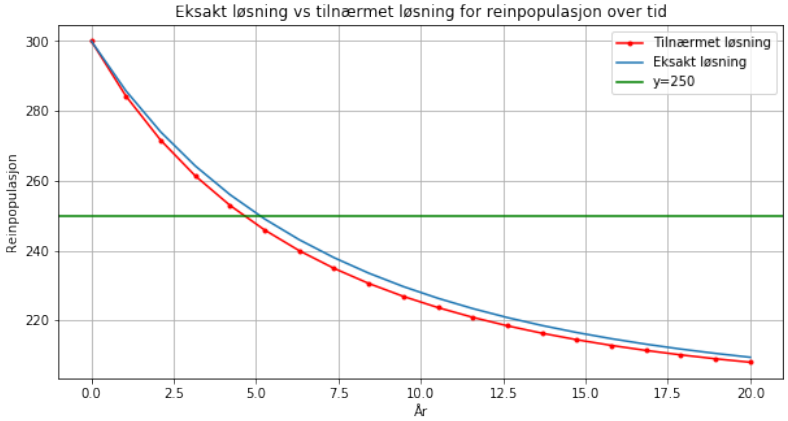
```python
'''
programmet finner utviklingen av reinpopulasjon over tid
'''

# importerer biblioteker
import numpy as np
from scipy.integrate import odeint
from sympy import solve
import matplotlib.pyplot as plt
from astropy.table import Table

# definerer en funksjon f(y, t) som returnerer vår funksjon
def f(y, t):
    dydt = 0.10*y * (1-y/200)
    return dydt

# lager en liste for t-verdier
t_verdier = np.linspace(0, 20, 20)

# initialverdi for y
y0 = 300

# kaller Eulers_metode() for å finne tilnærmede y-verdier
y_approx = Eulers_metode(f, y0, t_verdier)

'''
løser differensiallikningen ved å kalle på odeint()-funksjonen
parametere: funksjonen f(y, t), initialverdi y0, t_veridier
returnerer en liste av y-verdier (eksakt løsning)
'''
y_eksakt = odeint(f, y0, t_verdier)

'''
finner koeffisienter til en funksjon som passer y-verdier (velger polynom av grad 2)
koeffisientene er en nx1 matrise avhengig av polynomgraden
'''
koeff = np.polyfit(t_verdier, y_eksakt, 2)

'''
finner funksjonen y(t) ved å kalle på poly1d()-funksjonen
parametere: koeffisienter, variabel
returnerer funksjonen y(t)
'''
y = np.poly1d(koeff[:,0], variable='t')

# skriver ut funksjonen
print('y(t)=\n', y)

# setter størrelse for plottet
plt.figure(figsize=(10,5))

# setter tittel, og label for x- og y-aksen
plt.title("Eksakt løsning vs tilnærmet løsning for reinpopulasjon over tid")
plt.xlabel('År')
plt.ylabel('Reinpopulasjon')

# plotter grafen for den eksakte løsningen mot den tilnærmede løsningen
plt.plot(t_verdier, y_approx, 'r.-', label='Tilnærmet løsning')
plt.plot(t_verdier, y_eksakt, '-', label='Eksakt løsning')
plt.axhline(250, color='green', label='y=250')

# setter label for grafer
plt.legend()

# setter rutenett
plt.grid()

# viser plottet
plt.show()

# setter verdier i en tabell for å finne ut når reinpopulasjonen er 250
tabell = Table([np.round(t_verdier, decimals=0), np.round(y_eksakt, decimals=0), np.round(y_approx, decimals=0)],
               names=('År', 'Eksakt verdi', 'Tilnærmet verdi'))

# skriver ut tabellen
print(tabell)
```

Program A.45: Programmet finner utviklingen av reinpopulasjon over tid.

Output:

```
y(t)=
          2
0.2624 t - 9.271 t + 293.7
```

År	Eksakt verdi	Tilnærmet verdi
0.0	300.0	300.0
1.0	286.0	284.0
2.0	274.0	272.0
3.0	264.0	261.0
4.0	256.0	253.0
5.0	249.0	246.0
6.0	243.0	240.0
7.0	238.0	235.0
8.0	234.0	231.0
9.0	230.0	227.0
11.0	226.0	224.0
12.0	223.0	221.0
13.0	221.0	218.0
14.0	219.0	216.0
15.0	217.0	214.0
16.0	215.0	213.0
17.0	213.0	211.0
18.0	212.0	210.0
19.0	211.0	209.0
20.0	209.0	208.0



Figur A.22: Grafen viser utviklingen av reinpopulasjon over 20 år.

Vi ser både fra tabellen og fra grafen A.22 at tilnærmede verdier er litt forskjellige fra de eksakte verdier. Tabellen viser at reinpopulasjonen i år 5 er lik 249 blant de eksakte verdier, som er den nærmeste verdien til 250, mens blant de tilnærmede verdier er den nærmeste verdien til 250, i år 4 lik 253. Dersom vi øker antall punkter for t-verdier dvs. hvis vi velger Δt mindre, vil den nærmeste verdien for begge deler (tilnærmede og eksakte verdier) være i år 5.

"Reinpopulasjonen avtar raskt de første årene. Etter ca. 15 år, når populasjonen kommer under 215 rein, avtar den mye saktere.

Hvis vi plotter utviklingen av populasjonen over et bredere intervall, for eksempel de første 100 årene, ser vi at etter $40 - 50$ år begynner reinpopulasjonen å stabilisere seg på 200 rein, som er bæreevnen til stammen"(ProFag, 2021).

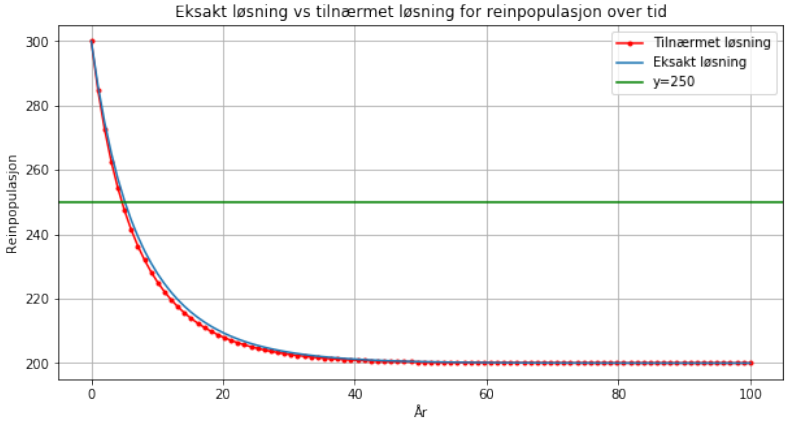
Vi endrer tidsintervallet, og løser likningen på nytt og ser at konklusjonen stemmer.

Output:

```
y(t)=
          2
0.01424 t - 1.841 t + 252.9
```

År	Eksakt verdi	Tilnærmet verdi
0.0	300.0	300.0
1.0	286.0	285.0
2.0	275.0	273.0
3.0	265.0	263.0
4.0	257.0	254.0
5.0	250.0	247.0
6.0	244.0	241.0
7.0	239.0	236.0
8.0	235.0	232.0
9.0	231.0	228.0
...	...	...
90.0	200.0	200.0
91.0	200.0	200.0
92.0	200.0	200.0
93.0	200.0	200.0
94.0	200.0	200.0
95.0	200.0	200.0
96.0	200.0	200.0
97.0	200.0	200.0
98.0	200.0	200.0
99.0	200.0	200.0
100.0	200.0	200.0

Length = 100 rows



Figur A.23: Grafen viser utviklingen av reinpopulasjon over 100 år.

Grafen A.23 viser at utviklingen av reinpopulasjon stabiliserer seg på $y = 200$ som er bæreevnen til reinpopulasjonen.

8 Matematikk S1

Kompetansemål etter matematikk S1

Mål for opplæringen er at eleven skal kunne

- anvende derivasjon til å analysere og forstå optimaliseringsproblemer
- bruke digitale verktøy til å simulere og utforske utfall i stokastiske forsøk, og forstå begrepet stokastiske variabler
- analysere et problem der sannsynlighet og kombinatorikk inngår, og bruke ulike strategier i problemløsingen

Kilde

8.1 Økonomiske optimeringsproblemer

Kostnadfunksjon En kostnadfunksjon er en funksjon for kostnader ved produksjon av x antall enheter av en vare. Vi viser en kostnadfunksjon med $K(x)$.

Grensekostnad Grensen som viser hvor mye koster å produsere én ekstra enhet ved en gitt produksjonsmengde.

Inntektsfunksjon En funksjon som viser inntekter ved salg av x antall enheter av en vare. Vi viser en inntektsfunksjon med $I(x)$.

Grenseinntekt Grensen som viser hvor stor inntekt én ekstra produsert enhet gir.

Overskuddsfunksjon Overskuddsfunksjonen viser hvor mye en bedrift har tjent ved produksjon og salg av x antall enheter. Vi viser en overskuddsfunksjon med $O(x)$. Overskuddet finner vi ved å trekke kostnader fra inntekter (Kristensen & Aanensen, 2018g):

$$inntekter - kostnader = overskudd \tag{A.31}$$

Overskudd kan være både positivt og negativt. Hvis overskuddet er positivt, sier vi at produksjon av varen er lønnsomt.

Oppgave En bedrift produserer og selger en vare. De totale kostnadene $K(x)$ kroner ved produksjon av x enheter av varen per dag er gitt ved:

$$K(x) = 0.01x^3 + 0.08x^2 + 10.25x + 3000, \quad x \in [0, 100] \tag{A.32}$$

Inntekten i kroner ved salg av x enheter av varen er:

$$I(x) = 550x - 5x^2, \quad x \in [0, 100] \tag{A.33}$$

a Ved hvilken produksjon vil kostnader og inntekter være like store?

b Undersøk om det lønner seg å øke produksjonen når $x = 50$.

c Bedriften vil tilpasse produksjonen slik at overskuddet $O(x)$ blir størst mulig. Bruk derivasjon til å finne den produksjonen som gir størst mulig overskudd per dag. Hvor stort er dette overskuddet?

Oppgaven er hentet fra (Kristensen & Aanensen, udatert, s. 48).

Løsning

a For å finne ut ved hvilken x-verdi er $K(x)$ og $I(x)$ like stor, setter vi de to likningene lik hverandre:

$$K(x) = I(x) \iff K(x) - I(x) = 0$$

Vi kaller denne funksjonen $f(x)$, og løser den ved hjelp av *fsolve()*-funksjonen fra *scipy.optimize*-modulen. Funksjonen tar funksjonen f og en startgjett x_0 som parametere og returnerer x-verdien til roten dvs. skjæringspunktet mellom de to funksjonene. x_0 kan være enten ett tall eller en n-dimensjonal *array* av flere tall (The-SciPy-community, 2021b).

Etter at vi løste oppgaven algebraisk, plotter vi grafen til de to funksjonene, samt skjæringspunktene mellom dem.

Algoritme

1 Importere biblioteket *Numpy*, modulen *matplotlib.pyplot*, og funksjonen *fsolve()* fra *scipy.optimize*-modulen.

2 Definere en funksjon $K(x)$ for kostnader; funksjonen tar x som parameter, og returnerer Kostnadfunksjonen A.32.

3 Definere en funksjon $I(x)$ for inntekter; funksjonen tar x som parameter, og returnerer Inntektsfunksjonen A.33.

4 Definere en funksjon $f(x)$, funksjonen tar x som parameter, og returnerer $I(x) - K(x)$.

5 Definere en variabel startgjett, vi setter denne variabelen lik en liste på intervall $[0, 100]$ med 2 punkter ved å kalle *linspace()*-funksjonen.

6 Finne x-koordinater til skjæringspunkter mellom K og I ved å kalle *fsolve()*-funksjonen. Funksjonen tar funksjonen $f()$ og startgjett x_0 som parametere, og returnerer x-koordinater til røtter. Vi lagrer resultatet i en variabel kalt x_1.

7 Sette x_1 i en av funksjonene I eller K og finne y-verdier.

8 Benytte en *for*-løkke som looper antall ganger lik lengde for startgjett-listen.

8.1 Skrive ut koordinatene til skjæringspunktene.

9 Definere og lage en liste for x-verdier ved å kalle *linspace()*-funksjonen; funksjonen tar startpunkt, sluttpunkt og antall punkter for plotting som parametere.

10 Definere og lage to sett av y-verdier ved å sette x-verdier i funksjonene for kostnad og inntekt.

11 Sette størrelse for plottet ved å kalle *figure()*-funksjonen fra *pyplot*. Vi definerer figurstørrelse som parameter til funksjonen:

```
figure(figsize=(x,y))
```

12 Sette tittel for plottet ved å kalle *title()*-funksjonen fra *pyplot*.

13 Sette label for x- og y-aksen ved å kalle henholdsvis *xlabel()*- og *ylabel()*-funksjonen.

14 Plotte grafer for kostnad og inntekt samt skjæringspunktene ved å bruke *plot()*-funksjonen.

15 Bruke en *for*-løkke sammen med *zip()*-funksjonen for å matche koordinatene to-og-to. Vi kaller også *around()*-funksjonen fra *NumPy*-biblioteket for å avrunde tallene; for dette bruker vi *np.around()* siden listene er av type *NumPy-array*. Syntaksen er som følger:

```
for i, j in zip(x, y)
```

15.1 Kalle *text()*-funksjonen fra *matplotlib.pyplot* for å sette tekst (koordinat) for punkter. Syntaksen er som følger:

```
text(i, j, '({},␣{})'.format(i, j))
```

16 Sette rutenett på plottet ved å kalle *grid()*-funksjonen.

17 Sette label for grafene ved å kalle *legend()*-funksjonen; funksjonen tar en liste av strenger som parameter.

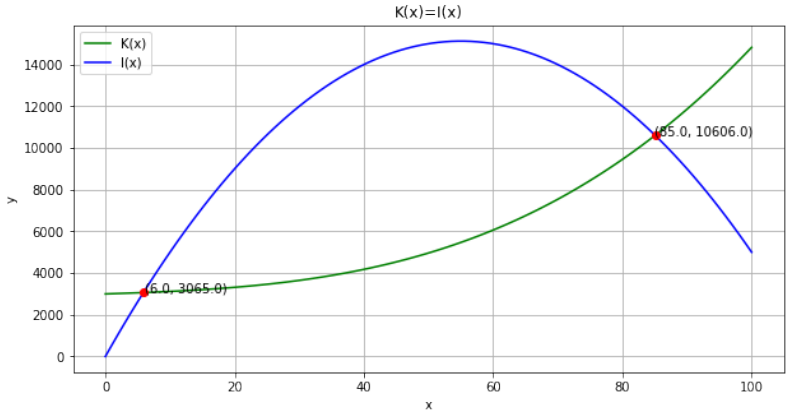
18 Vise plottet ved å kalle *show()*-funksjonen.

```python
# a: finne produksjon x som har like store kostnader og inntekter

# importerer biblioteker
from scipy.optimize import fsolve
import matplotlib.pyplot as plt
import numpy as np

# definerer en funksjon K(x) for kostnader
def kostnad(x):
    return 0.01*x**3+0.08*x**2+10.25*x+3000

# definerer en funksjon I(x) for inntekter
def inntekt(x):
    return -5*x**2 + 550*x

'''
funksjon f(x) er definert ved K(x) = I(x)
parameter: x
returnerer K(x) - I(x)
'''
def f(x):
    return inntekt(x) - kostnad(x)

x0 = np.linspace(0, 100, 2) # definerer startgjett
x_1 = fsolve(f, x0)         # finner x-verdier til skjæringspunkt mellom K og I
y_1 = kostnad(x_1)          # setter x_1 i K(x) (eller I(x)) og finner y_1

# skriver ut koordinater til skjæringspunkter
for i in range(len(x0)):
    print(f'({x_1[i]}, {y_1[i]})')

x = np.linspace(0, 100, 100) # definerer x_verdier

# definerer y_verdier for K og I
y_verdier1 = kostnad(x)
y_verdier2 = inntekt(x)

plt.figure(figsize=(10,5)) # setter størrelse for plottet
plt.title('K(x)=I(x)')     # setter tittel for plottet

# setter x- og y-label for aksene
plt.xlabel('x')
plt.ylabel('y')

# plotter funksjoner K(x) og I(x) samt skjæringspunkter
plt.plot(x, y_verdier1, 'g', x, y_verdier2, 'b', x_1, y_1,'ro')
'''
setter koordinat for punktene
zip()-funksjonen matcher to lister to-og-to
around()-funksjonen avrunder tallene,
vi bruker np.around() siden listene av type NumPy-array
text()-funksjonen setter tekst for punkter
'''
for i,j in zip(np.around(x_1, decimals=0), np.around(y_1, decimals=0)):
    plt.text(i, j, '({}, {})'.format(i, j))

plt.grid()                    # setter rutenett for plottet
plt.legend(['K(x)', 'I(x)']) # setter label for grafer
plt.show()                    # viser plottet
```

Program A.46: Programmet finner produksjon x som har like store kostnader og inntekter.

Output:

```
(5.888228629025245, 3065.1695640255384)
(85.0637605961929, 10605.851494073984)
```



Figur A.24: Illustrasjon for Produksjon av x enheter når $K(x) = I(x)$.

Vi ser både fra output av programmet og grafen A.24 at ved en produksjon på 6 enheter og 85 enheter vil inntektene og kostnadene være like store.

b Det er lønnsomt å øke produksjonen til x enheter dersom grenseinntektene $I'(x)$ er større enn grensekostnadene $K'(x)$. For å sjekke om det lønner seg å øke produksjonen når $x = 50$, sjekker vi dermed om $I'(50) > K'(50)$

Algoritme

1 Importere bibliotekene *Numpy* og *SymPy*, og modulen *matplotlib.pyplot*.

2 Definere x som symbol ved å kalle *Symbol()*-funksjonen fra *SymPy*-biblioteket. *Symbol* er den viktigste klassen i *SymPy*-biblioteket. Vi bruker denne klassen for symbolske beregninger (SymPy-Development-Team, 2021c).

3 Derivere $K(x)$ og $I(x)$ ved å kalle *diff()*-funksjonen fra *SymPy*-biblioteket.

4 Oversette derivert-uttrykk til *NumPy*-funksjon ved å kalle *lambify()*-funksjonen; dette trenger vi for å kunne mate den deriverte med tall, og beregne y-verdi for en gitt x-verdi. Funksjonen tar en variabel og funksjon som parametere, og returnerer en funksjon av type *SymPy*-funksjon (SymPy-Development-Team, 2021a).

5 Beregne $K'(50)$ og $I'(50)$, og skrive ut resultatet.

6 Definere og lage en liste for x-verdier på intervallet $[0, 100]$ ved å kalle *linspace()*-funksjonen.

7 Definere og lage y-verdier for $K'(x)$ og $I'(x)$ ved å mate funksjoner med x-verdier.

8 Sette størrelse for plottet ved å kalle *figure()*-funksjonen fra *pyplot*. Vi definerer figurstørrelse som parameter til funksjonen:

```
figure(figsize=(x,y))
```

9 Sette tittel for plottet ved å kalle *title()*-funksjonen fra *pyplot*.

10 Sette label for x- og y-aksen ved å kalle henholdsvis *xlabel()*- og *ylabel()*-funksjonen.

11 Plotte grafer for $K'(x)$ og $I'(x)$ samt skjæringspunktene ved å bruke *plot()*-funksjonen.

12 Sette y-verdi (koordinat) for punktene ved å kalle *annotate()*-funksjonen fra *Pyplot*-modulen. Funksjonen tar tekst, og x og y koordinater for annotasjon (Hunter mfl., 2021c). Syntaksen for funksjonen er som følger:

```
annotate(str(f(x)),xy=(x,  f(x)))
```

13 Sette rutenett på plottet ved å kalle *grid()*-funksjonen.

14 Sette label for grafene ved å kalle *legend()*-funksjonen; funksjonen tar en liste av strenger som parameter.

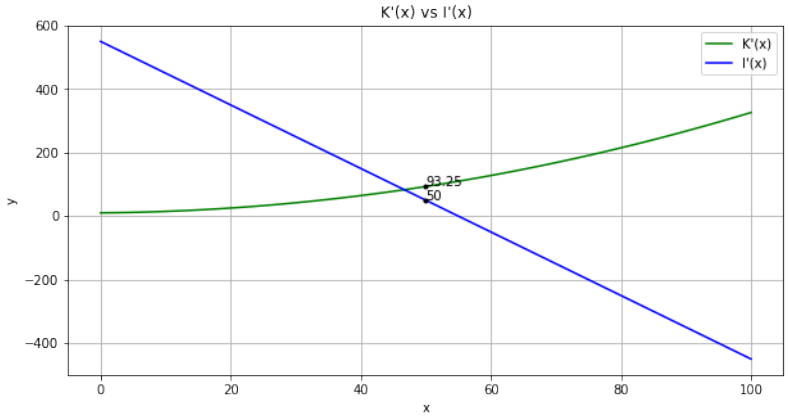
15 Vise plottet ved å kalle *show()*-funksjonen.

```
# b: programmet sjekker lønnsomheten for å øke produksjon når x=50

# importerer biblioteker
from sympy import *
import numpy as np
import matplotlib.pyplot as plt

# definerer x som symbol ved å kalle Symbol()-funksjonen
x = Symbol('x')

# deriverer K(x) og I(x) ved å kalle diff()-funksjonen
K_derivert = kostnad(x).diff(x)
I_derivert = inntekt(x).diff(x)

# oversetter derivert-uttrykk til SymPy-funksjon
K_prime = lambdify(x, K_derivert)
I_prime = lambdify(x, I_derivert)

# beregner K'(50) og I'(50)
print(f'K\'(50): {K_prime(50)}\n\
I\'(50): {I_prime(50)}')

x = np.linspace(0, 100, 100)  # definerer x_verdier

# definerer y_verdier for K' og I'
y_verdier1 = K_prime(x)
y_verdier2 = I_prime(x)

plt.figure(figsize=(10,5))     # setter størrelse for plottet
plt.title('K\'(x) vs I\'(x)')  # setter tittel for plottet

# setter x- og y-label for aksene
plt.xlabel('x')
plt.ylabel('y')

# plotter funksjoner K'(x) og I'(x) samt skjæringspunkter
plt.plot(x, y_verdier1, 'g', x, y_verdier2, 'b', 50, K_prime(50), 'k.', 50, I_prime(50), 'k.')

# setter y-verdi for punkter på plottet
plt.annotate(str(K_prime(50)),xy=(50, K_prime(50)))
plt.annotate(str(I_prime(50)),xy=(50, I_prime(50)))

plt.grid()                          # setter rutenett for plottet
plt.legend(['K\'(x)', 'I\'(x)'])    # setter label for grafer
plt.show()                          # viser plottet
```

Program A.47: Programmet sjekker lønnsomheten for å øke produksjon når x=50.

Output:

```
K'(50): 93.25
I'(50): 50
```



Figur A.25: Grafen viser grensekostnaden vs grenseinntekten når $x = 50$.

Grafen A.25 viser at grensekostnaden er betydelig høyere enn grenseinntekten når $x = 50$. Derfor vil det ikke lønne seg å øke produksjonen når $x = 50$.

c Overskudd er lik *inntekt* − *kostnad*. Vi har allerede definert funksjonen $f(x)$ i deloppgave *a* gitt ved $I(x) - K(x)$, men for å ha mer oversiktlig programkode, definerer vi en ny funksjon kalt $O(x) = I(x) - K(x)$. Så finner vi den deriverte til $O(x)$, og setter den lik null for å finne ekstremal punkter for $O(x)$. Overskuddet vil være størst på toppunktet av $O(x)$.

Algoritme

1 Importere bibliotekene *Numpy* og *SymPy*, og modulen *matplotlib.pyplot*.

2 Definere funksjonen $O(x)$; funksjonen tar x som parameter og returnerer $I(x) - K(x)$.

3 Definere x som symbol ved å kalle *Symbol()*-funksjonen.

4 Finne den deriverte av $O(x)$ ved å kalle *diff()*-funksjonen.

5 Skrive ut $O'(x)$.

6 Sette $O'(x) = 0$ for å finne x-koordinat av ekstremalpunkter. Vi kaller *solve()*-funksjonen for å gjøre dette.
Funksjonen tar en funksjon f og en (eller flere) variabel som parametere, og løser funksjonen med hensyn til den variabelen (SymPy-Development-Team, 2021b).

7 Konverter x-verdiene til ekstremalpunkter til *Numpy*-array, og sette dem i $O(x)$ for å finne y-verdier av ekstremalpunkter.

8 Skrive ut resultatet.

9 Kalle *lambify()*-funksjonen for å oversette derivert-uttrykket for $O(x)$ til *NumPy*-funksjon; funksjonen tar x og en funksjon (her $O'(x)$ som parametere.

10 Finne dobbeltderiverte av $O(x)$; vi skal sjekke om ekstremalpunktet er et toppunkt.

11 Skrive ut $O''(x)$.

12 Oversette uttrykket for den dobbeltderiverte av O til *NumPy*-funksjon ved å kalle *lambify()*-funksjonen.

13 Finne y-verdi for den positive roten av $O''(x)$ ved å sette inn x-koordinat til ekstremalpunktet i $O''(x)$.

14 Skrive ut resultatet.

15 Definere og lage en liste for x-verdier ved å kalle *linspace()*-funksjonen; vi velger en bredere rekkevidde enn $[0, 100]$ fordi verdiene vi skal vise på plottet er store tall.

16 Finne y-verdier for $O(x)$, $O'(x)$, og $O''(x)$.

17 Sette størrelse for plottet ved å kalle *figure()*-funksjonen fra *pyplot*. Vi definerer figurstørrelse som parameter til funksjonen:

```
figure(figsize=(x,y))
```

18 Sette tittel for plottet ved å kalle *title()*-funksjonen fra *pyplot*.

19 Sette label for x- og y-aksen ved å kalle henholdsvis *xlabel()*- og *ylabel()*-funksjonen.

20 Plotte funksjoner $O(x)$, $O'(x)$, $O''(x)$, og ekstremalpunktene ved å kalle *plot()*-funksjonen.

21 Bruke en *for*-løkke sammen med *zip()*-funksjonen for å matche koordinatene to-og-to. Syntaksen er som følger:

```
for i, j in zip(x, y)
```

21.1 Kalle *text()*-funksjonen fra *matplotlib.pyplot* for å sette tekst (koordinat) for punkter. Syntaksen er som følger:

```
text(i, j, '({}, {})'.format(i, j))
```

22 Sette rutenett på plottet ved å kalle *grid()*-funksjonen.

23 Sette label for grafene ved å kalle *legend()*-funksjonen; funksjonen tar en liste av strenger som parameter.

24 Vise plottet ved å kalle *show()*-funksjonen.

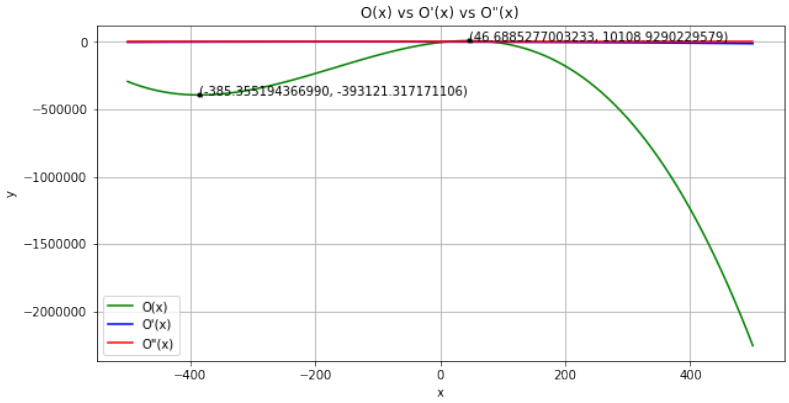
25 Skrive ut ved hvilket x-verdi er overskuddet størst.

```python
# c: programmet finner det største overskuddet

# importerer biblioteker
from sympy import *
import numpy as np
import matplotlib.pyplot as plt

# definerer en funksjon for overskudd
def O(x):
    return inntekt(x) - kostnad(x)

# definerer x som symbol ved å kalle Symbol()-funksjonen
x = Symbol('x')

# finner den deriverte av O(x) ved å kalle diff()-funksjonen
O_derivert = O(x).diff(x)
print(f'O\'(x) = {O_derivert}')      # skriver ut O'(x)

# setter O'(x)=0 for å finne (x-koordinat til) ekstremalpunkter
ekst_punkter = solve(O_derivert)

# finner y-koordinat for ekstremalpunkter
y_ekst_punkter = O(np.array(ekst_punkter))
print(f'x-koordinat av ekstremalpunkter for O\'(x): {ekst_punkter}') # skriver ut resultatet

# oversetter derivert-uttrykk til NumPy-funksjon
O_prime = lambdify(x, O_derivert, 'numpy')

O_andrederivert = O_prime(x).diff(x)             # finner O"(x)
print(f'O\"(x) = {O_andrederivert}')             # skriver ut O"(x)

O_2prime = lambdify(x, O_andrederivert, 'numpy') # oversetter derivert-uttrykk til NumPy-funksjon
y_2prime = O_2prime(ekst_punkter[1])             # finner y-verdi for O"(positiv x-koordinat)
print(f'O\"({ekst_punkter[1]}) = {y_2prime}')    # skriver ut resultatet

x = np.linspace(-500, 500, 100)                  # definerer x_verdier

# definerer y_verdier for O(x), O'(x) og O"(x)
y_1 = O(x)
y_2 = O_prime(x)
y_3 = O_2prime(x)

plt.figure(figsize=(10,5))             # setter størrelse for plottet
plt.title('O(x) vs O\'(x) vs O\"(x)')  # setter tittel for plottet

# setter x- og y-label for aksene
plt.xlabel('x')
plt.ylabel('y')

# plotter funksjoner O(x), O'(x), O"(x), og ekstremalpunkter
plt.plot(x, y_1, 'g', x, y_2, 'b', x, y_3, 'r', ekst_punkter, y_ekst_punkter,'k.')

# setter koordinat for punkter
for i,j in zip(ekst_punkter, y_ekst_punkter):
    plt.text(i, j, '({}, {})'.format(i, j))

plt.grid()                                 # setter rutenett for plottet
plt.legend(['O(x)', 'O\'(x)', 'O\"(x)']) # setter label for grafer
plt.show()                                 # viser plottet

'''
siden vi fikk negativ verdi for O"(+ekstremalpunkt),
har bedriften størst overskudd for denne x-verdien.
skriver ut resultatet
'''
print(f'Bedriften har størst overskudd når x = {round(ekst_punkter[1])};\
 overskuddet er {round(y_ekst_punkter[1])} kr.')
```

Program A.48: Programmet finner det største overskuddet.

Output:

```
O'(x) = -0.03*x**2 - 10.16*x + 539.75
x-koordinat av ekstremal punkter for O'(x): [-385.355194366990, 46.6885277003233]
O''(x) = -0.06*x - 10.16
O''(46.6885277003233) = -12.9613116620194

Bedriften har størst overskudd når x = 47; overskuddet er 10109 kr.
```



Figur A.26: Grafen viser når bedriften har størst overskudd.

Slik output av programmet viser, og vi ser fra grafen A.26, vil bedriften ha størst overskudd ved $x = 47$ enheter; overskuddet ved denne verdien er omtrent 10109 kr. per dag.

8.2 Sannsynlighet og kombinatorikk

8.2.1 Sannsynlighet Det er vanlig å buke urnemodeller ved introduksjon av sannsynlighetsregning. I programmering av urnemodeller kan vi bruke datastrukturen *liste* for representasjon av en urne.

For eksempel anta at vi har tre svarte og to hvite kuler i en urne, da kan vi lage en liste på denne måten:

```
urne_liste = ['Svart', 'Hvit', 'Svart', 'Hvit', 'Svart']
```

Dersom vi skal lage en modell med tilbakelegging, trenger vi ikke å gjøre noen endringer på urnelisten underveis i kjøring av programmet. I modellering av trekkesituasjoner uten tilbakelegging, må vi fjerne elementer som trekkes ut, fra listen.

For at modellen vi lager, gir oss en god empirisk verdi, velger vi å foreta så mange trekninger som for eksempel 100 trekninger. Vi bruker *randint()*-funksjonen for å simulere trekningen fra urnen (Bueie, 2019).

Oppgave I en urne er det tre svarte kuler og sju hvite. I en annen urne er det tre hvite kuler og sju svarte. Det skal trekkes ti kuler uten tilbakelegging, fem fra hver urne, og legges opp i en tredje urne.

Lag et program som viser hvor stor andel hvite kuler og hvor stor andel svarte kuler det er mest sannsynlig at det havner i den tredje urnen.

Oppgaven er hentet fra (Bueie, 2019, s. 117).

Løsning Programmet er en modifisert versjon av løsningsforslaget for oppgaven fra boka «Programmering for matematikklærere» (Bueie, 2019, s. 188).

Programmet løser oppgaven og simulerer hvor stor andel hvite kuler og hvor stor andel svarte kuler det er mest sannsynlig å havne i vår tredje urne etter 100 trekninger.

Algoritme

1 Importere modulene *Random* og *Matplotlib.pyplot*.

2 Definere en variabel *teller* som teller antall forsøk, og sette den lik 0.

3 Definere en variabel *antall_trekninger* for å telle antall trekninger, og sette den lik 0.

4 Definere en variabel *antall_svart* for å telle antall svarte kuler som trekkes ut, og sette den lik 0.

5 Bruke en *while*-løkke som looper så lenge telleren er mindre eller lik 100.

- 5.1 Lage to liste av strenger som modellerer urnene.
- 5.2 Lage en tom liste for å sette elementene som trekkes ut inn i den.
- 5.3 Benytte en *for*-løkke som looper 5 ganger, denne løkken er for å trekke ut 5 elementer fra hver urne-liste.
  - 5.3.1 Generere et tilfeldig heltall ved å kalle *randint()*-funksjonen; funksjonen tar start- og slutt-verdi på det intervallet som tallet skal genereres på, som parameter. Lagre det genererte tallet i en variabel kalt *indeks_1*.
  - 5.3.2 Bruke det tilfeldige tallet som indeks i urne-listen 1, og lagre elementet ved indeksen i en variabel kalt *trekning_1*.
  - 5.3.3 Fjerne det elementet fra urne-listen 1 ved å kalle *pop()*-funksjonen. Funksjonen tar indeks som parameter, og sletter elementet ved indeksen fra listen.
  - 5.3.4 Generere et annet tilfeldig heltall, og lagre det genererte tallet i en variabel kalt *indeks_2*.
  - 5.3.5 Bruke det tilfeldige tallet som indeks i urne-listen 2, og lagre elementet ved indeksen i en variabel kalt *trekning_2*.
  - 5.3.6 Fjerne det elementet fra urne-listen 2.
  - 5.3.7 Legge de to trekningene i urne-listen 3, ved å kalle *extend()*-funksjonen; funksjonen tar en liste av elementer som parameter og setter dem inn i urne-listen 3.
- 5.4 Bruke en *for*-løkke som looper 10 ganger.
  - 5.4.1 Inkrementere antall trekninger med 1.
  - 5.4.2 Sjekke om det finnes noen svarte kuler i urne-listen 3, hvis ja
    - 5.4.2.1 inkrementere antall svarte med 1.
- 5.5 Inkrementere telleren (antall forsøk) med 1.

6 Finne andelen for svarte kuler ved å dele antall svarte på totalt antall trekninger.

7 Finne andelen for hvite kuler ved å trekke andelen for de svarte kulene fra 1 (siden den største verdien for sannsynlighet er 1.).

8 Multiplisere andelen for svarte kuler med 100 for å finne sannsynligheten for svarte kuler.

9 Multiplisere andelen for hvite kuler med 100 for å finne sannsynligheten for hvite kuler.

10 Skrive ut resultatet.

11 Lage data for simulering av trekninger; definere 4 tomme lister for x- og y-koordinater for hvite og svarte kuler.

12 Finne antall hvite kuler ved å trekke antall svarte fra totalt antall trekninger.

13 Bruke en *for*-løkke som looper antall ganger lik antall svarte kuler.

- 13.1 Generere et tilfeldig flyttall mellom 0 og 1 ved å kalle *random()*-funksjonen; lagre det tilfeldige tallet i en variabel kalt *x_1*.
- 13.2 Generere et tilfeldig flyttall mellom 0 og 1 ved å kalle *random()*-funksjonen; lagre det tilfeldige tallet i en variabel kalt *y_1*.
- 13.3 Legge *x_1* til listen for x-koordinater til svarte kuler.
- 13.4 Legge *y_1* til listen for y-koordinater til svarte kuler.

14 Bruke en *for*-løkke som looper antall ganger lik antall hvite kuler.

- 14.1 Generere et tilfeldig flyttall mellom 0 og 1 ved å kalle *random()*-funksjonen; lagre det tilfeldige tallet i en variabel kalt *x_2*.
- 14.2 Generere et tilfeldig flyttall mellom 0 og 1 ved å kalle *random()*-funksjonen; lagre det tilfeldige tallet i en variabel kalt *y_2*.
- 14.3 Legge *x_2* til listen for x-koordinater til hvite kuler.
- 14.4 Legge *y_2* til listen for y-koordinater til hvite kuler.

15 Sette størrelse for plottet ved å kalle *figure()*-funksjonen fra *pyplot*. Vi definerer figurstørrelse som parameter til funksjonen:

```
figure(figsize=(x,y))
```

16 Sette tittel for plottet ved å kalle *title()*-funksjonen fra *pyplot*.

17 Definere et subplott ved å kalle *gca()*-funksjonen fra *pyplot*-modulen; kalle subplottet *ax*.

18 Sette bakgrunnsfarge for subplottet ved å kalle *set_facecolor()*-funksjonen fra *matplotlib.pyplot* (*ax*); funksjonen tar farge med streng datatype som parameter.

19 Plotte svarte kuler ved å kalle *scatter()*-funksjonen; funksjonen tar to lister av x- og y-verdier som parametere.

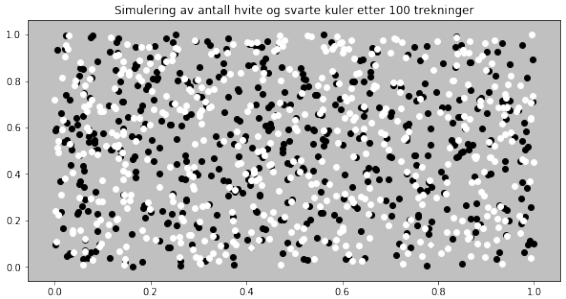
20 Plotte hvite kuler ved å kalle *scatter()*-funksjonen; funksjonen tar to lister av x- og y-verdier som parametere.

```
'''
programmet er en modifisert versjon av løsningsforslaget fra boka "programmering for matematikklærere"
'''
# importerer modulene random og matplotlib.pyplot
import random
import matplotlib.pyplot as plt

teller = 0              # definerer en variabel for å telle antall forsøk
antall_trekninger = 0   # definerer en variabel for å telle antall trekninger
antall_svart = 0        # definerer en variabel for å telle antall svarte kuler som trekkes ut

# bruker while-løkke som looper 100 ganger, i hver runde trekkes 10 kuler, 5 fra hver urne-liste
while teller <= 100:

    # lager liste for urne_lister (i hver runde lager vi listene på nytt)
    urne_liste_1 = ['Svart', 'Svart', 'Svart', 'Hvit', 'Hvit', 'Hvit', 'Hvit', 'Hvit', 'Hvit', 'Hvit']
    urne_liste_2 = ['Hvit', 'Hvit', 'Hvit', 'Svart', 'Svart', 'Svart', 'Svart', 'Svart', 'Svart', 'Svart']
    urne_liste_3 = []  # lager en tom urne_liste for å sette trekninger i

    for i in range(5): # for-loop for å trekke 5 kuler fra hver urne
        '''
        genererer et tall som indeks til et element i urne-liste 1 ved å kalle på randint()-funksjonen
        randint()-funksjonen tar to parametere start og stop i et intervall der begge er inkludert
        i intervallet, og genererer et tilfeldig tall
        '''
        indeks_1 = random.randint(0, len(urne_liste_1) - 1)
        trekning_1 = urne_liste_1[indeks_1]                 # trekker elementet ut fra urne-liste 1
        urne_liste_1.pop(indeks_1)                          # fjerner elementet fra urne-liste 1

        # genererer et tall som indeks til et element i urne-liste 2
        indeks_2 = random.randint(0, len(urne_liste_2) - 1)
        trekning_2 = urne_liste_2[indeks_2]                 # trekker elementet ut fra urne-liste 2
        urne_liste_2.pop(indeks_2)                          # fjerner elementet fra urne-liste 2

        # legger trekninger i urne-liste 3,extend()-funksjonen brukes for å legge elementer i en liste
        urne_liste_3.extend([trekning_1, trekning_2])

    # bruker for-løkke for å telle antall elementer av hver farge i urne-liste 3
    for j in range(10):
        antall_trekninger += 1                          # inkrementerer antall trekninger
        if urne_liste_3[j] == 'Svart':                  # sjekker om element på indeks j er en svart kule
            antall_svart += 1                           # og hvis ja, inkrementerer antall svarte kuler
    teller += 1                                         # inkrementerer antall forsøk

andel_svart =  antall_svart / antall_trekninger # finner andelen av svarte kuler fra totale trekninger
andel_hvit = 1 - andel_svart                # andelen av hvite kuler er 1 - andelen av de svarte kulene
svart_sannsynlighet = andel_svart * 100 # beregner sannsynligheten av svarte og hvite kuler i prosent
hvit_sannsynlighet = andel_hvit * 100

# skriver ut resultatet
print(f'Sannsynlighet for at det havner flest svarte kuler i den tredje urnen er {svart_sannsynlighet:.2f}')
print(f'Sannsynlighet for at det havner flest hvite kuler i den tredje urnen er {hvit_sannsynlighet:.2f}')

svart_liste_x = []  # lager x- og y-liste for hvite og svarte kuler
svart_liste_y = []
hvit_liste_x = []
hvit_liste_y = []
antall_hvit = antall_trekninger - antall_svart  # finne antall hvite kuler

# lager tilfeldige koordinater for svarte kuler, random() returnerer et tilfeldig float tall mellom 0 og 1
for i in range(antall_svart):
    x_1 = random.random()
    y_1 = random.random()
    svart_liste_x.append(x_1) # legger x og y koordinatene i lister for x- og y-verdier for svarte kuler
    svart_liste_y.append(y_1)

for i in range(antall_hvit):  # lager tilfeldige koordinater for hvite kuler
    x_2 = random.random()
    y_2 = random.random()
    hvit_liste_x.append(x_2)  # legger x og y koordinatene i lister for x- og y-verdier for svarte kuler
    hvit_liste_y.append(y_2)

plt.figure(figsize=(10,5))                                          # setter størrelse for plottet
plt.title('Simulering av antall hvite og svarte kuler etter 100 trekninger') # setter tittel for plottet

ax = plt.gca()  # bruker gca()-funksjonen for å plotte svarte og hvite punkter i samme plott
ax.set_facecolor('silver')  # setter background color for plottet
ax.scatter(hvit_liste_x, hvit_liste_y, color="black") # plotter svarte og hvite punkter
ax.scatter(svart_liste_x, svart_liste_y, color="white")
```

Program A.49: Programmet finner sannsynligheten for andelen hvite og svarte kuler i den tredje urnen.

Output:

```
Sannsynlighet for at det havner flest svarte kuler i den tredje urnen er 49.80
Sannsynlighet for at det havner flest hvite kuler i den tredje urnen er 50.20
```



Figur A.27: Simulering for andelen av hvite og svarte kuler.

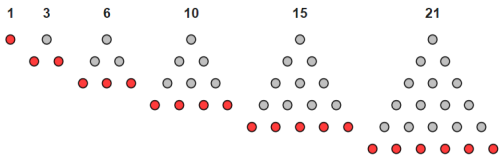
Modellen (se figur A.27) viser at sannsynligheten for andelen av hvite og svarte kuler som vil havne i den tredje urnen er nesten like stor. Selv om i vårt tilfelle andelen for hvite kuler er større enn svarte kuler, er forskjellen uansett ikke så stor. Hvis vi kjører programmet flere ganger, vil det hende at andelen for svarte kuler øker. Dette er noe som stemmer med virkeligheten også siden vi har like stor hvite og svarte kuler totalt sett.

8.2.2 Kombinatorikk

"Trekanttallet T_n er et figurantall som kan representeres i form av et trekantet rutenett av noder der den første raden inneholder et enkelt element og hver påfølgende rad inneholder ett mer element enn det forrige"(Weisstein, 2002d). De trekanttallene er derfor:

$\Delta_1 \to 1$
$\Delta_2 \to 1 + 2 = 3$
$\Delta_3 \to 1 + 2 + 3 = 6$
$\Delta_4 \to 1 + 2 + 3 + 4 = 10$
.
.
.
$\Delta_n \to 1 + 2 + 3 + 4 + (n-1) + n = \frac{1}{2}n(n+1)$

La T_n være trekanttall nummer n. Vi ser at hvert trekanttall T_n er summen av det forrige trekanttallet T_{n-1} pluss n prikker:



Figur A.28: Figuren viser de fem første trekanttallene.

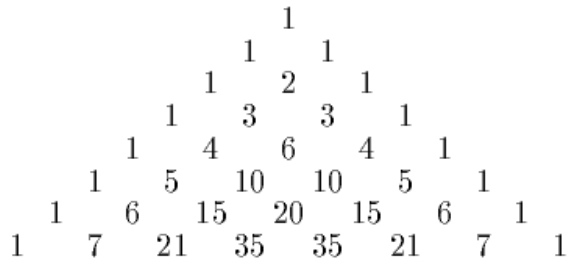
Source: Wikipedia

Vi kan derfor skrive den rekursive formelen for trekanttallet T_n gitt ved:

$$T_n = T_{n-1} + n \tag{A.34}$$

Pascals talltrekant Pascals talltrekant er et trekant utvalg av binomialkoeffisienter som brukes i sannsynlighets teori, kombinatorikk og algebra.

Hver rad i talltrekantet starter og slutter med tallet 1, og de andre tallene som ligger inn i trekanten er lik summen av de to nærmeste oppstående tallene. Figur A.29 viser illustrasjonen av Pascals talltrekant.



Figur A.29: Pascals talltrekant

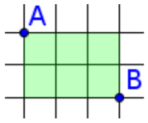
Source: Wikipedia

"Det viser seg at tallene i Pascals talltrekant forteller hvor mange ≪korteste veier≫ som leder fra toppen og fram til et krysningspunkt i talltrekanten"(Kristensen & Aanensen, 2018e).

Oppgave Gatebildet i sentrum av Kristiansand, kvadraturen, er regelmessig bygd opp med rette gater hvor gater som krysser hverandre danner vinkler på omtrent 90 grader. ≪Kvartalene≫, områdene avgrenset av gater, har tilnærmet form av rektangler.

Vi tenker oss nå byen enda mer regelmessig slik at alle ≪kvartaler≫ har en kvadratisk grunnflate.

Tenk deg at du skal gå fra gatehjørne A til gatehjørne B.

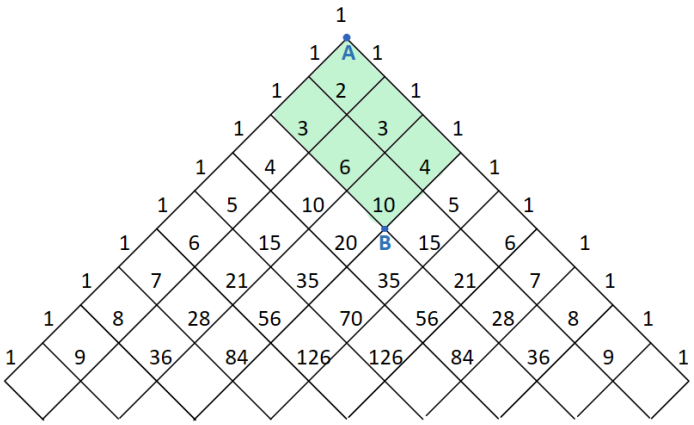


Source: NDLA

Hvor mange forskjellige ≪korteste veier≫ er det mellom A og B?

Oppgaven er hentet fra (Kristensen & Aanensen, 2018e).

Løsning Vi bruker Pascals talltrekant for å løse oppgaven. Hvis vi setter figuren gitt i oppgaven på Pascals talltrekant slik at punkt A ligger på toppunktet i trekanten, og hver node i figuren ligger på tilsvarende tallet i trekanten, vil trekanttallet som matcher punkt B vise antall mulige korteste veier.



Figur A.30: Kvadratisk Pascals talltrekant

Som sagt hver node i Pascals talltrekant, tilsvarer en binomialkoeffisient. Derfor kan vi bruke formelen for antall permutasjoner for å finne den n-te noden i Pascals talltrekantet:

$$\binom{n}{k} = \frac{n!}{k!(n-1)!} \tag{A.35}$$

Vi kan tilpasse formelen med problemstillingen i oppgaven, og løse den. Anta n er totalt antall noder til den korteste veien fra A til B, og k er lik antall noder enten til bredden eller til lengden av rektangelet (se på den grønne rektangelet i figur A.30) avhengig av hvilken vei man ønsker å ta.

Algoritme

1 Importere *factorial()*-funksjonen fra *math*-modulen.

2 Definere en funksjon kalt *finn_antall_korteste_veier()*; funksjonen tar n og k som parametere.

- 2.1 Definere en variabel *antall* for å lagre resultatet i; sette variabelen lik 0.
- 2.2 Bruke *try-catch*-uttrykk for å unngå eventuelle feil-verdier.
- 2.3 Prøve å
  - 2.3.1 finne antall veier ved å bruke formelen A.35. Vi benytter operatoren *heltallsdivisjon* for å få heltall som resultat.
- 2.4 I unntatte tilfeller med feil-verdier
  - 2.4.1 returneres 0.
- 2.5 Returnere antall korteste veier.

3 Definere funksjonen *main()* som klientprogram.

- 3.1 Be brukeren om å oppgi tallene n og k; konvertere input-verdiene til heltall.
- 3.2 Finne antall korteste veier ved å kalle funksjonen *finn_antall_korteste_veier()*; funksjonen tar n og k som parametere.
- 3.3 Skrive ut resultatet.

4 Kalle *main()*-funksjonen.

```
'''
Programmet finner antall korteste veier fra punkt A til B vha. binomialkoeffisienter
'''

# importerer biblioteker
from math import factorial as fac
import scipy.special

'''
funksjon for å finne antall mulige korteste vei fra A til B ved hjelp av Pascals talltrekant
parametere: totalt antall noder n, antall noder til bredde eller lengde
return: antall mulige korteste vei
'''
def finn_antall_korteste_veier(n, k):

    # definere en variable antall
    antall = 0

    # bruker formelen for binomial koeffisient, try-except vil hjelpe med å unngå eventuelle feil
    try:
        antall = fac(n) // fac(k) // fac(n - k)
    except ValueError:
        antall = 0

    # kan også kalle på binom()-funksjonen fra scipy-biblioteket
    # antall = scipy.special.binom(n,k)

    # returnerer resultatet
    return antall

# klient-programmet
def main():

    # får n og k verdier fra brukeren
    n = int(input('Skriv inn totalt antall noder fra A til B: '))
    k = int(input('Hvilken vei du ønsker å ta? Skriv inn antall noder til veien: '))

    # finner antall korteste veier ved å kalle finn_antall_korteste_veier()-funksjonen
    antall = finn_antall_korteste_veier(n, k)

    # skriver ut resultatet
    print(f'Fra punkt A til B, finnes det {antall} korteste veier.')

main()
```

Program A.50: Programmet finner antall korteste veier fra punkt A til B vha. binomialkoeffisienter.

Output:

```
Skriv inn totalt antall noder fra A til B: 5
Hvilken vei du ønsker å ta? Skriv inn antall noder til veien: 2
Fra punkt A til B, finnes det 10 korteste veier.
```

9 Matematikk S2

Kompetansemål etter matematikk S2

Mål for opplæringen er at eleven skal kunne

- utforske egenskaper ved ulike rekker og gjøre rede for praktiske anvendelser av egenskaper ved rekker
- utforske rekursive sammenhenger ved å bruke programmering og presentere egne framgangsmåter
- modellere og analysere eksponentiell og logistisk vekst i reelle datasett
- forstå begrepene forventningsverdi, varians og standardavvik, og bruke disse størrelsene til å tolke stokastiske variabler

Kilde

9.1 Følger og rekker

9.1.1 Fibonacci-tallfølgen

"Fibonacci-tallene F_n er leddene for sekvensen

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...$$

hvor hvert ledd er summen av de to forrige leddene, som begynner med verdiene $F_0 = 0$, og $F_1 = 1$"(Falcon & Plaza, 2007, s. 38).

Formelt kan a_n-leddet i Fibonacci-tallfølgen uttrykkes som følger:

$$a_n = a_{n-1} + a_{n-2} \tag{A.36}$$

Oppgave Lag et program som finner tall nummer n i følgen.

Programkoden for Fibonacci-tallfølgen er lånet fra (Hafting & Ljosland, 2014).

Løsning For å finne ledd nummer n, trenger vi som nevnt i oppgaven, de to tidligere leddene i tallfølgen. Dermed bruker programkoden to lokalvariabler *temp_1* og *temp_2* som hjelpevariabler, og lagrer summen av de to som *resultat*; altså:

$$resultat = temp_1 + temp_2$$

Tabell A.5 viser illustrasjon av tankegangen:

n	$ledd_{n-2}$	$ledd_{n-1}$	$ledd_n$
	$temp_1$	$temp_2$	resultat
0			0
1			1
2	0	1	1
3	1	1	2
4	1	2	3
5	2	3	5
6	3	5	8
.			
.			
.			
10	21	34	55

Tabell A.5: Fibonacci tallfølgen

Algoritme

1 Definere en funksjon kalt *fibonacci()*, funksjonen tar ledd-nummer som parameter.

1.1 Definere hjelpevariabler *temp_1* og *temp_2*, og sette dem lik henholdsvis 0 og 1; disse er altså startverdier for å finne ledd-nummer 2, og fortsette med å finne de neste tallene i følgen.

1.2 Definere en variabel *resultat* for å lagre resultatet av beregningen.

1.3 Bruke en *for*-løkke som looper n ganger.

1.3.1 Sette *temp_1* lik *temp_2*.

1.3.2 Sette *temp_2* lik *resultat*.

1.3.3 Sette *resultat* lik summen av *temp_1* og *temp_2*.

1.4 Returnere *resultat*.

2 Definere *main()*-funksjonen som klientprogram.

2.1 Spørre brukeren om hvilket ledd hun ønsker å finne i Fibonacci tallfølgen; konvertere input-verdien til heltall.

2.2 Finne leddet ved å kalle *fibonacci()*-funksjonen.

2.3 Skrive ut resultatet.

3 Kalle *main()*-funksjonen.

```
'''
funksjonen finner n-te ledd i Fibonacci tallfølgen.
metoden tar ledd-indeksen som parameter, og returnerer verdien til leddet
'''
def fibonacci(n):

    # definerer to hjelpevariabel for å finne ledd nummer n; tilordner temp_1 -> ledd_0 og temp_2 -> ledd_1
    temp_1 = 0
    temp_2 = 1

    # definerer en variabel resultat for å lagre summen av de to tidligere leddene inn i
    resultat = 0

    # for-løkke for å finne leddet
    for i in range(n):

        '''
        algoritmen for å finne ledd_n:

        tem_1 + temp_2 = resultat
                 ||        ||
        ->      temp_1 + temp_2 = resultat
                          ||      ||
                ->       temp_1 + temp_2 = resultat
        '''
        temp_1 = temp_2
        temp_2 = resultat
        resultat = temp_1 + temp_2

    # returnerer resultatet
    return resultat

# klient-programmet
def main():

    # får ledd-indeksen fra brukeren, konverterer den til heltall
    n = int(input('Skriv inn indeksen til leddet: '))

    # finner leddet med å kalle fibonacci()-funksjonen
    ledd_n = fibonacci(n)

    # skriver ut resultatet
    print(f'Ledd nummer {n} i Fibonacci tallfølgen er {ledd_n}')

main()
```

Program A.51: Programmet finner n-te ledd i Fibonacci tallfølgen.

Eksempel output:

```
Skriv inn indeksen til leddet: 10
Ledd nummer 10 i Fibonacci tallfølgen er 55
```

9.2 Eksponential vekst

Oppgave Du skal nå lage en forenklet modell for harepopulasjonen på fjellet. I et avgrenset område på fjellet settes det ut N_0 harer. Anta at populasjonsveksten er gitt ved:

$$N(t) = N_0 * 1.4^t \tag{A.37}$$

Lag en funksjon som tar inn to argumenter: antall år gitt ved t og antall harer vi starter med gitt ved N_0, og finner harepopulasjonen.

Plott deretter harepopulasjonen de første x årene. Hvorfor er modellen urealistisk?

Oppgaven er hentet fra (ProFag, 2021).

Løsning Vi definerer først en funksjon *beregn_harepopulasjon()*; funksjonen tar inn de to verdiene som er bedt i oppgaven som parametere, og returnerer harepopulasjonen.

Deretter definerer vi en annen funksjon for å plotte harepopulasjonen ved hjelp av denne funksjonen.

Algoritme

1 Importere *NumPy*-biblioteket, og *matplotlib.pyplot*-modulen.

2 Definere en funksjon *beregn_harepopulasjon()*; funksjonen tar antall harer ved tiden 0, og antall år som parametere.

- 2.1 Beregne harepopulasjonen N_t ved å bruke formelen A.37.
- 2.2 Returnere N_t.

3 Definere en funksjon *plot_harepopulasjon()* for å plotte grafen.

- 3.1 Få antall harer ved tiden 0, N_0, og antall år t fra brukeren; konvertere input-verdiene til heltall.
- 3.2 Beregne harepopulasjonen N_t ved tiden t ved å kalle *beregn_harepopulasjon()*-funksjonen.
- 3.3 Skrive ut resultatet.
- 3.4 Lage en liste for x-verdier ved å kalle funksjonen *linspace()* fra *Numpy*-biblioteket; funksjonen tar startpunkt, sluttpunkt og antall punkter for plotting som parametere.
  Her setter vi startverdi lik 0 og sluttverdi lik t. Antall punkter kan være hva som helst.
- 3.5 Lage en liste for y-verdier ved å kalle *beregn_harepopulasjon()*-funksjonen, funksjonen tar startverdi N_0 og x-verdier som parametere.
- 3.6 Sette tittel for plottet ved å kalle *title()*-funksjonen fra *pyplot*.
- 3.7 Sette label for x- og y-aksen ved å kalle henholdsvis *xlabel()*- og *ylabel()*-funksjonen.
- 3.8 Sette rutenett på plottet ved å kalle *grid()*-funksjonen.
- 3.9 Tegne grafen ved å kalle *plot()*-funksjonen; funksjonen tar x- og y-verdier som parametere.

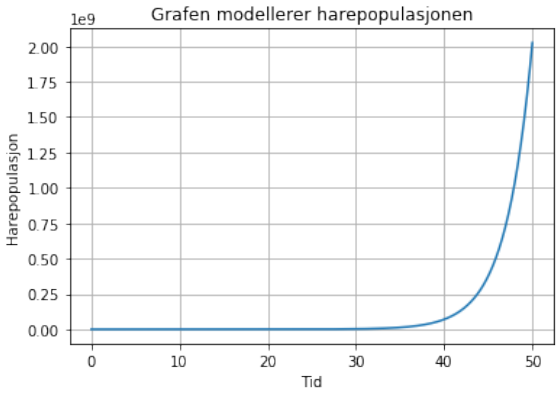
4 Kalle *plot_harepopulasjon()*-funksjonen.

```python
'''
Programmet beregner harepopulasjonen N ved tiden t og tegner grafen til funksjonen
'''

# importerer biblioteker
import numpy as np
import matplotlib.pyplot as plt

'''
funksjonen beregner harepopulasjon
parameter: antall harer N_0 ved tiden 0, og antall år t
return: harepopulasjon ved tiden t
'''
def beregn_harepopulasjon(N_0, t):

    # beregner populasjon med bruk av formelen
    N_t = N_0 * 1.4**t

    # returnerer resultatet
    return N_t

# funksjonen for plotting
def plot_harepopulasjon():

    # Får startpopulasjon og antall år fra brukeren
    N_0 = int(input('Skriv inn antall harer ved tiden 0: '))
    t = int(input('Skriv inn antall år: '))

    # beregner harepopulasjonen
    N_t = beregn_harepopulasjon(N_0, t)

    # skriver ut harepopulasjonen
    print(f'Harepopulasjonen etter {t} år er {N_t}.')

    # lager en liste av x-verdier med hjelp av linspace()-funksjonen
    x_verdier = np.linspace(0, t, 1000)

    # beregner y-verdi for hver x-verdi med å kalle funksjonen beregn_harepopulasjon()
    y_verdier = beregn_harepopulasjon(N_0, x_verdier)

    # setter tittel for grafen
    plt.title("Grafen modellerer harepopulasjonen")

    # setter navn for x-aksen
    plt.xlabel("Tid")

    # setter navn for y-aksen
    plt.ylabel("Harepopulasjon")

    # lager rutenett i plottet
    plt.grid()

    # plotter grafen med å kalle plot-funksjonen() (fra pylab)
    plt.plot(x_verdier, y_verdier)

plot_harepopulasjon()
```

Program A.52: Programmet beregner harepopulasjonen N ved tiden t og tegner grafen til funksjonen.

Eksempel output:

```
Skriv inn antall harer ved tiden 0: 100
Skriv inn antall år: 50
Harepopulasjonen etter 50 år er 2024891623.9764307.
```



Figur A.31: Grafen viser utviklingen av harepopulasjonen som en eksponential funksjon.

"Grafen A.31 viser en eksponential vekst av harer. Grunnen til at modellen er urealistisk er fordi den ikke tar hensyn til andre faktorer. En økende harepopulasjon ville ført til mangel på mat siden vi ser på et avlukket område. Det ville også ført til en økende populasjon av rovdyr, siden rovdyrene får mer mat. I et avlukket område ville altså harene hatt en begrenset bæreevne"(ProFag, 2021).

9.3 Statistikk

Statistikk hjelper oss med å undersøke store mengder av *enheter*. Ved statistiske undersøkelser er vi interessert i en *populasjon* av enheter, og i stedet for å undersøke hele populasjonen, velger vi ut noen få. Ut fra resultatet vi får fra dette *utvalget*, forsøker vi å konkludere noe om enhetene i hele populasjonen (Løvås, 2018).

I statistikk har vi to typer data:

- Diskrete data: "Diskrete data er observasjoner av diskrete variabler. Her er bare enkelte tall langs talls linjen aktuelle som kjennetegn"(Løvås, 2018, s. 41).
- Kontinuerlige data: "Kontinuerlige data er observasjoner av kontinuerlige variabler. Her kan alle tallverdier innen et gitt intervall brukes for å angi et kjennetegn"(Løvås, 2018, s. 42).

Vi jobber ofte med store datasett når vi undersøker et fenomen i en populasjon. Datasettet kan inneholde mange forskjellige verdier, og dette fører til kompleksitet i analysen av datasettet. Det finnes dermed metoder som hjelper oss med å holde oversikt over datasettet, og beregne nøkkeltall som beskriver dataene.

Frekvensfordeling En frekvensfordeling er en grafisk eller tabellarisk representasjon som viser antall observasjoner innen et gitt intervall. Størrelsen på intervallet er avhengig av dataene som analyseres, og analytikerens mål. Frekvensfordeling brukes vanligvis innenfor en statistisk sammenheng. Generelt kan frekvensfordeling være assosiert med kartleggingen av en normalfordeling (Young, 2020).

Histogram "Et histogram er et søylediagram som visualiserer innholdet i en frekvenstabell. Histogrammet er definert slik at arealet av alle søylene til sammen er lik 1"(Løvås, 2018, s. 44).

Sentralmål Sentralmål er en representativ målingsverdi i datasettet som befinner seg et sted sentralt i histogrammet (Løvås, 2018).

- Modus: Den verdien som forekommer flest ganger i datasettet (Løvås, 2018).
- Median: Den verdien som ligger i midten av et sortert datasett (Løvås, 2018).
- Utvalgets gjennomsnitt: Verdien lik summen av alle verdier delt på antall verdier i datasettet, og er gitt ved følgende formel (Løvås, 2018):

$$\bar{x} = \frac{x_1 + x_2 + ... + x_n}{n} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{A.38}$$

Spredningsmål Spredningsmål viser variasjonen i datasettet:

- Variasjonsbredde: Differansen mellom størst observasjon og minst observasjon, dvs. variasjonsbredden beskriver bredden på utvalgets histogram (Løvås, 2018).
- Utvalgets standardavvik: "Standardavvik er et typisk avvik fra gjennomsnitts verdien. Målingen tar utgangspunkt i å se hvor mye hver enkelt verdi avviker i forhold til gjennomsnittet"(Løvås, 2018, s. 57). Formelen for standardavvik er gitt ved:

$$s_x = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{A.39}$$

- Utvalgets varians: Hovedideen bak varians er at vi kvadrerer standardavvik, og finner noe som kalles avviks kvadrat. Den gjennomsnittlige verdien av avviks kvadratene er lik utvalgets varians, og er gitt ved:

$$s_x^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 \tag{A.40}$$

I Python kan vi enkelt bruke NumPy-biblioteket for å beregne sentralmål og spredningsmål for et datasett. (For modus bruker vi *mode()*-funksjonen fra *scipy.stats*-modulen.)

```
'''
Programmet beregner statistiske målinger
'''

# importerer biblioteker
import random
from numpy import *
from scipy import stats

# definerer en tom liste
en_liste = []

# looper for å generere 20 tilfeldige tall
for i in range(20):

    # kaller i hver loop randint()-funksjonen som genererer heltall på intervallet [0, 100]
    random_tall = random.randint(0,100)

    # legger tallet til listen
    en_liste.append(random_tall)

# skriver ut listen
print(en_liste, '\n')

# sentralmål:

# kaller på mode()-funksjonen fra stats-modulen
modus = stats.mode(en_liste)

# kaller følgende funksjoner fra Numpy-biblioteket
median = median(en_liste)
gjennomsnitt = average(en_liste)
gjennomsnitt_2 = mean(en_liste)

# spredningsmål

# kaller på ptp()-funksjonen fra NumPy-biblioteket
variasjonsbredde = ptp(en_liste)

# kaller på var()- og std()-funksjonen

# varians for populasjon
varians = var(en_liste, ddof=0)

# varians for et utvalg
utvalgetsvarians = var(en_liste, ddof=1)

# standardavvik for populasjon
standardavvik = std(en_liste, ddof=0)

# standardavvik for et utvalg
utvalgetsstandardavvik = std(en_liste, ddof=1)

# skriver ut resultatet
print(f'Sentralmål:\n\
        Modus: {modus}\n\
        Median: {median:.2f}\n\
        Gjennomsnitt beregnet med average()-funksjonen: {gjennomsnitt:.2f}\n\
        Gjennomsnitt beregnet med mean()-funksjonen: {gjennomsnitt_2:.2f}\n')

print(f'Spredningsmål:\n\
        Variasjonsbredde: {variasjonsbredde}\n\
        Varians for populasjon: {varians:.2f}\n\
        Utvalgets varians: {utvalgetsvarians:.2f}\n\
        Standardavvik for populasjon: {standardavvik:.2f}\n\
        Utvalgets standardavvik: {utvalgetsstandardavvik:.2f}')
```

Program A.53: Programmet viser funksjoner for å finne sentralmål og spredningsmål for et datasett i Python.

Eksempel output:

```
[22, 27, 20, 56, 61, 72, 63, 75, 67, 55, 25, 20, 96, 79, 49, 73, 79, 25, 69, 11]

Sentralmål:
        Modus: ModeResult(mode=array([20]), count=array([2]))
        Median: 58.50
        Gjennomsnitt beregnet med average()-funksjonen: 52.20
        Gjennomsnitt beregnet med mean()-funksjonen: 52.20

Spredningsmål:
        Variasjonsbredde: 85
        Varians for populasjon: 611.26
        Utvalgets varians: 643.43
        Standardavvik for populasjon: 24.72
        Utvalgets standardavvik: 25.37
```

Noen oppmerksomheter om statistiske funksjoner i Python

- Forskjellen mellom *average()*-funksjonen og *mean()*-funksjonen er at vi kan beregne vektet gjennomsnitt ved å buke *average()*. Vektet gjennomsnitt er den verdien vi får fra multiplisering av hvert element i listen med en faktor som gjenspeiler dens betydning. Dersom vi ikke bruker vektfaktor for beregning av gjennomsnittet, fungerer *average()* lik som *mean()*-funksjonen.
- Siden *NumPy*-biblioteket ikke har noe innebygd funksjon for beregning av modus, beregner vi modus ved å benytte *mode()*-funksjonen fra *scipy.stats*-modulen.

Oppgave Ved en skole ble høyden til alle elevene på Vg2 målt. Resultatet er presentert i tabellen under:

Høyde til elevene

Høyde i cm	Frekvens
$[150, 160\rangle$	6
$[160, 165\rangle$	21
$[165, 170\rangle$	60
$[170, 175\rangle$	73
$[175, 180\rangle$	64
$[180, 185\rangle$	67
$[185, 190\rangle$	24
$[190, 200\rangle$	8
Sum	323

Finn søylehøyden (histogramhøyden) i hvert intervall, og tegn histogram som illustrerer resultatet.

Løsning Histogramhøyden finner vi ved å dele frekvens på klassebredde:

$$histogramhoyde = \frac{frekvens}{klassebredde} \tag{A.41}$$

Søylehøyden viser altså antall elever per centimeter. Vi får igjen frekvensen hvis vi multipliserer søylehøyden med klassebredde.

Algoritme

1 Importere biblioteket *NumPy*, og modulen *matplotlib.pyplot*.

2 Lage en liste av frekvenser. Vi velger å lage liste med *array()*-funksjonen fra NumPy. NumPy-arrayer er mer fleksible og gir oss mulighet for å utføre operasjoner som subtrahering og divisjon av to lister på samme måte som vi gjør med enkelte tall.

3 Lage to lister ut fra intervaller; en liste består av startverdier på hvert intervall, og den andre inkluderer sluttverdier på hvert intervall. Vi skal beregne klassebredde for intervaller for å finne søylehøyde. Dessuten vil vi ha behov for de to listene for plotting av histogrammer.

4 Definere en liste *klassebredde* ved å subtrahere startverdi-listen fra sluttverdi-listen.

5 Beregne søylehøyden ved å dele frekvens-listen på klassebredde-listen.

6 Skrive ut listen.

7 Definere et plott (figure) som består av to subplotter *ax1* og *ax2*. Plottet er et 1×2 plott, det betyr at plottene skal ligge horisontalt, og ved siden av hverandre. Vi definerer også størrelse for plottet. Syntaksen er som følger:

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(x, y))
```

8 Sette tittel for plottet (figuren) ved å kalle *suptitle()*-funksjonen.

9 Sette label for x- og y-aksen for subplottene ved å kalle *set()*-funksjonen; funksjonen tar *xlabel* og *ylabel* som parametere. Syntaksen for, f. eks. *ax1* er som følger:

```
ax1.set(xlabel='xlabel', ylabel='ylabel')
```

Merk at *ax1* her er den samme som *plot*. Vi har (som ble nevnt) definert et subplott, men funksjonaliteten er den samme som plott.)

10 Plotte diagrammer (histogrammer) ved hjelp av *bar()*-funksjonen. (og ikke *hist()*). Grunnen er at vi har allerede frekvenser. *hist()*-funksjonen brukes når vi ikke har telt frekvenser i datasettet. *hist()*-funksjonen finner automatisk frekvenser, og dette er veldig effektivt når vi jobber med store datasett.

bar()-funksjonen tar følgende parametere (Nelson, 2018):

```
bar(x, height, width=0.8, bottom=None, ec='k', align='center')
```

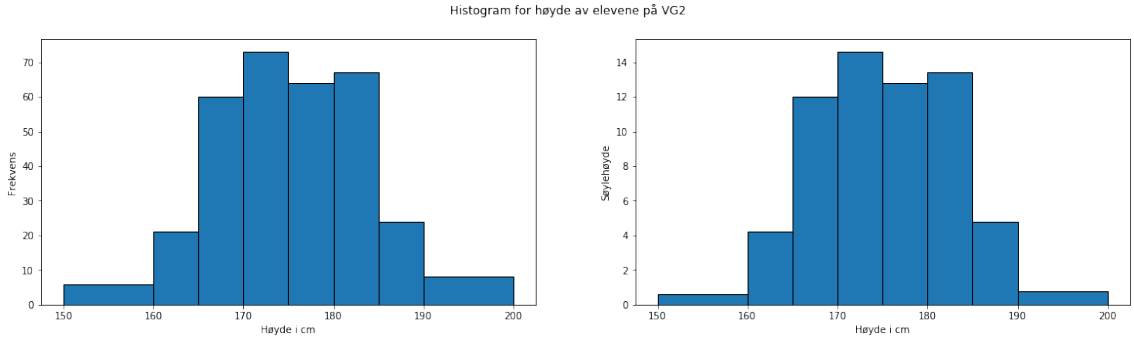
- **x:** x-koordinat for hver stolpe; i denne oppgaven setter vi x lik startverdier i hvert intervall.
- **height:** Høyden for hver stolpe; her setter vi frekvens for plott 1 og søylehøyde for plott 2 som høyde.
- **widht:** Bredde til intervaller (sluttverdi - startverdi).
- **bottom:** y-koordinat for hver stolpe, og som standard er lik 0.
- **ec:** ec står for «edge color», og setter farge rundt stolpene. I denne oppgaven velger vi 'k' som står for svartfarge.
- **align:** Justerer stolpene med x-koordinat, og kan settes enten som 'center' eller 'edge'. Hvis man velger 'center', vil x-koordinater settes i midten av intervallet. I denne oppgaven velger vi 'edge' som setter x-koordinater til venstre.
  For å justere stolpene på høyre siden, setter vi *width* negativt, og velger 'edge'.

```
'''
Programmet tegner histogram for et datasett
'''

# importerer biblioteker
import matplotlib.pyplot as plt
import numpy as np

# lager liste av frekvensverdier
frekvens = np.array([6, 21, 60, 73, 64, 67, 24, 8])

# startpunkter i hvert intervall
min_liste = np.array([150, 160, 165, 170, 175, 180, 185, 190])

# sluttpunkter i hvert intervall
max_liste = np.array([160, 165, 170, 175, 180, 185, 190, 200])

# lager en liste for klassebredde
klassebredde = max_liste-min_liste

# finner søylehøyden for hvert intervall
søylehøyde = frekvenser / klassebredde

# skriver ut resultatet
print(f'søylehøyden i hvert intervall: {list(søylehøyde)}')

# definere en 1x2 figur for plotting
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 5))

# setter title for figuren
fig.suptitle('Histogram for høyde av elevene på Vg2')

# setter x- og y-label for plotter
ax1.set(xlabel='Høyde i cm', ylabel='Frekvens')
ax2.set(xlabel='Høyde i cm', ylabel='Søylehøyde')

# plott1: frekvenser mot klasser(intervaller)
ax1.bar(min_liste, frekvens, width=klassebredde, ec="k", align="edge")

# plott2: søylehøyde mot klasser(intervaller)
ax2.bar(min_liste, søylehøyde, width=klassebredde, ec="k", align="edge")
```

Program A.54: Programmet finner søylehøyden, og tegner histogram for datasettet.

Output:

```
søylehøyden i hvert intervall: [0.6, 4.2, 12.0, 14.6, 12.8, 13.4, 4.8, 0.8]
```



Figur A.32: Histogrammet til høyre illustrerer høyden vs frekvenser; og histogrammet til venstre illustrerer høyden vs søylehøyden.

10 Tillegg

10.1 Numeriske metoder

"Numerisk matematikk handler om å lage algoritmer for å tilnærme matematiske løsninger med minst mulig feil"(Haraldsrud mfl., 2020).

Vi kan løse matematiske problemer som for eksempel derivasjon og integrasjon med hjelp av numeriske metoder. Disse matematiske metoder er spesielt viktig når vi ønsker å løse mer realistiske problemstillinger som ikke lar seg å løses så enkelt med analytisk matematikk, eller de ikke har en eksakt løsning i det hele tatt. Numeriske metoder spiller også en sentral rolle for å finne nullpunkter i diskrete data og store datasett (Haraldsrud mfl., 2020).

10.1.1 Halverings metode

Halverings metode tar utgangspunkt i skjæringssetningen der vi har en kontinuerlig funksjon f, og to punkter a og b på x-aksen. Dersom $f(a)$ og $f(b)$ har forskjellige fortegn, finnes det ett (eller flere) nullpunkt c på intervallet $[a, b]$.

I halverings metoden finner vi først et midtpunkt m gitt ved:

$$m = \frac{a+b}{2} \tag{A.42}$$

Hvis $f(m)$ og $f(a)$ har forskjellige fortegn, lager vi et nytt intervall $[a, b]$ der $b = m$. Hvis $f(m)$ og $f(b)$ har forskjellige fortegn, lager vi intervallet $[a, b]$ der $a = m$. Vi repeterer, og finner midtpunkter helt til $f(m) = 0$.

10.1.2 Newtons metode

Newtons metode finner et tilnærmet nullpunkt til en funksjon f ved hjelp av nullpunktet til den n-te tangenten til funksjonen.

Vi bruker først en startgjett x_0, og deretter finner nullpunktet x_{n+1} til den n-te tangenten ved å ta i bruk nullpunktet x_n til den forrige tangenten. Vi gjentar prosessen helt til $f(x_{n+1}) = 0$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{A.43}$$

10.1.3 Eulers metode

Eulers metode er spesielt brukelig for å løse første ordens differensiallikninger. Vi kan også løse høyere ordens differensiallikninger ved å bruke Eulers metoden flere ganger.

Gitt en initialtilstand $f(x)$, og en differensiallikning som angir $f'(x)$, kan vi finne en tilnærmet verdi til $f(x+1)$:

$$f(x+1) = f(x) + f'(x).\Delta x \tag{A.44}$$

Der Δx er steglengde, og regnes utfra et intervall $[a, b]$ med antall steglengde n:

$$\Delta x = \frac{b-a}{n} \tag{A.45}$$

10.1.4 Rektangelmetoden Vi kan bruke *Riemann-summene* for å beregne integraler numerisk. Summen beregnes ved å dele området under grafen i geometriske former som rektangler, trapeser, paraboler eller kubikk. Deretter summerer vi arealet av de geometriske formene. Den endelige summen er en numerisk tilnærming for en bestemt integral.

Riemann-summene har forskjellige typer avhengig av beregning av x_i-verdier:

- Venstre Riemann-sum
- Høyre Riemann-sum
- Midtpunktmetoden

Når vi finner en tilnærmet verdi for en bestemt integral ved å summere et antall rektangler, kaller vi metoden for *rektangelmetoden*.

Rektangelmetoden (venstre Riemann-sum), tar venstre x-verdier i et intervall $[a, b]$, og beregner arealet av de n venstre rektangler som ligger under en graf f.

Det bestemte integralet av en funksjon $f(x)$ over intervallet $[a, b]$ kan tilnærmes ved arealet av n rektangler med bredden gitt ved:

$$h = \frac{b-a}{n} \tag{A.46}$$

Formelen for metoden er gitt ved (Haraldsrud mfl., 2020):

$$\int_a^b f(x)\,dx \approx h \sum_{k=1}^{n} f(x_i) \tag{A.47}$$

10.2 Følger og rekker

10.2.1 Aritmetiske tallfølger En tallfølge der differansen mellom et ledd og leddet foran er konstant, kalles en aritmetisk tallfølge. Differansen mellom to ledd som følger etter hverandre i en aritmetisk tallfølge, er gitt ved:

$$d = a_n - a_{(n-1)} \quad n > 1$$

En rekursiv formel for en aritmetisk tallfølge blir derfor:

$$a_n = a_{(n-1)} + d \tag{A.48}$$

I en aritmetisk tallfølge er tall nummer n gitt ved formelen:

$$a_n = a_1 + (n-1)d \tag{A.49}$$

Hvor a_n er n-te ledd i følgen, a_1 er det første leddet og d er konstant (Kristensen & Aanensen, 2018c).

10.2.2 Geometriske tallfølger En tallfølge der forholdet mellom et ledd og leddet foran er konstant, kalles en geometrisk tallfølge. I en geometrisk tallfølge kan vi alltid finne neste ledd i tallfølgen ved å multiplisere med kvotienten k. Den rekursive formelen for en geometrisk tallfølge blir derfor:

$$a_n = a_(n-1).k \tag{A.50}$$

I en geometrisk tallfølge er ledd nummer n gitt ved formelen (Kristensen & Aanensen, 2018c):

$$a_n = a_1.k^{(n-1)} \tag{A.51}$$

10.2.3 Aritmetiske rekker "Når vi adderer leddene i en tallfølge, får vi en tallrekke"(Kristensen & Aanensen, 2018d).

Tallrekker eller bare rekker er enten *endelige* eller *uendelige*. Summen av de n første leddene i en rekke vises med S_n.

"Når vi adderer leddene i en aritmetisk tallfølge, får vi en aritmetisk rekke"(Kristensen & Aanensen, 2018d).

Summen av de n første leddene i en aritmetisk rekke er gitt ved:

$$S_n = \frac{a_1 + a_n}{2}.n \tag{A.52}$$

10.2.4 Geometriske rekker "Når vi adderer leddene i en geometrisk tallfølge, får vi en geometrisk tallrekke. I en geometrisk rekke er forholdet mellom et ledd og det foregående leddet konstant. Vi kaller dette forholdstallet for rekkens kvotient, k"(Kristensen & Aanensen, 2018d).

Summen av de n første leddene i en geometrisk rekke er gitt ved:

$$S_n = a_1.(\frac{k^n - 1}{k - 1}) \quad k \neq 1 \tag{A.53}$$

Dersom $k = 1$, er:

$$S_n = n.a_1 \tag{A.54}$$

B Visjonsdokument

1 Innledning

Dette dokumentet oppgir visjonen for bacheloroppgaven som gjennomføres i forbindelse med faget TDAT3001 Bacheloroppgave Dataingeniør våren 2021. Hensikten med oppgaven er å finne ut hvilke matematiske problemer egner seg best for å øke dybdeforståelse, engasjement og kreativitet hos elever, og anbefalte fremgangsmåter for implementasjon av disse. Hensikten med dette prosjektet er å utforske bruk av programmering i matematikk, og utvikle matematiske algoritmer og programkode som støtter matematikklærere i programmerings undervisning på den norske videregående skolen.

2 Sammendrag problem og produkt

2.1 Problemsammendrag

Problem med	å anvende programmering i matematikk, og finne relevante materialer
berører	lærere i tjenesten
som resultatet av dette	vil de ikke være i stand til å bruke programmering til å øke fagforståelsen i matematikk. Konsekvensen er at de mister motivasjon og ser ikke helt poenget med å bruke tid på å lære seg programmering og dermed gir opp.
En vellykket løsning vil	være å tilby lærere algoritmer og programkoder for noen matematiske oppgaver som er relevant til læreplaner for matematikkfagene på videregående skoler.

2.2 Produktsammendrag

For	matematikklærere i tjenesten
som	har behov for matematisk modellering, algoritmer, og programkoder som hjelpemiddel i undervisning av matematikkfag på videregående skole,
produktet navngitt	≪ProMat≫
som	er en samling av matematiske oppgaver samt algoritmer og programkoder til løsninger. Oppgavene blir valgt slik at de samsvarer de nye læreplaner og gir dybdelæring og forståelse i faget.
I motsetning til	de eksisterende løsninger som ikke viser stegene for å konvertere problemet til et program,
har vårt produkt	som mål at lærere og elever blir mer engasjert i programmering og kan samtidig forbedre fagforståelsen/dybdelæring (i egne fag).

3 Overordnet beskrivelse av interessenter og brukere

3.1 Oppsummering interessenter

Navn	Utdypende beskrivelse	Rolle under utviklingen
NTNU-AIT	NTNU-AIT har innført et prosjektbasert opplæringskurs for lærere i tjenesten slik at de kan anvende programmering i sine fag. I slutten av kurset skal deltakere gjennomføre et prosjekt hvor de skal vise hvordan de vil bruke sin kunnskap i programmering i praksis, og på sin læreplan.	Evaluerer
Lærere i tjenesten	Lærere som deltar i dette kurset har forskjellige kunnskapsnivå i programmering. De fleste av dem har ikke hatt programmering fra før, og det kan være både utfordrende og tidkrevende for dem å finne ut hvordan de skal anvende programmering i sine fag. Resultatet av dette prosjektet vil tilbys til dem, testes av dem og eventuelt kan de bruke det i sine læringsplaner.	Tester

3.2 Oppsummering brukere

Navn	Utdypende beskrivelse	Rolle under utviklingen	Representert av
Lærere i tjenesten	Lærere i tjenesten er sluttbrukere av produktet	Bruker	NTNU-AIT

3.3 Brukermiljøet

Resultatet av dette prosjektet skal spesielt brukes av lærere i tjenesten som underviser matematikk på videregående skole. Etter at oppdragsgiveren har evaluert resultatet av prosjektet, og testet det, vil resultatet benyttes av lærere i tjenesten hvem som underviser matematikk på videregående skole.

3.4 Sammendrag av brukernes behov

Behov	Prioritet	Påvirker	Dagens løsning	Foreslått løsning
Matematisk modellering, algoritmer, og programkoder	Høy	Matematikkfag	Tradisjonelle løsninger	Pseudokoder/algoritmer og programkoder som viser fremgangsmåte for løsninger

3.5 Alternativer til vårt produkt

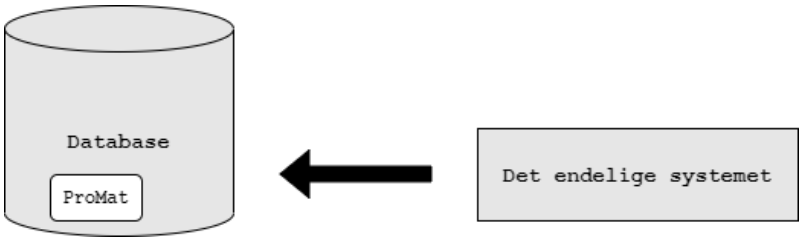
Det eksisterer allerede noen nettsider som tilbyr etterutdannings kurs til lærere, og opplæring for elever for å tilpasse seg til de nye læreplaner etter fagfornyelsen fra høsten 2020. Eksempel på slike nettsider er som følger:

- Campus Matte: Campus Matte er utviklet til fagfornyelse med vekt på dybdelæring og tilpasset opplæring. Link
- ProFag – realfaglig programmering: ProFag er etterutdanning for lærere i PROgrammering for FAGenes skyld. Fagene er naturfag, biologi, kjemi, fysikk og matematikk. Tilbudet er spesielt rettet mot realfagslærere med tanke på fagfornyelsen 2020. Link
- kikora: Slik det står på nettsiden, følger Kikora de nye læreplaner, og tilbyr opplæring for inklusiv programmering, GeoGebra og CAS for elever på alle trinn. Link
- FLIPCLASS: De tilbyr videoopplæring i Python Programmering i forbindelse med fagfornyelsen fra høsten 2020 både for elever og lærere. Link
- Programmering i matematikk: etterutdannings kurs for lærere i VGS. Link

4 Produktoversikt

4.1 Produktets rolle i brukermiljøet

ProMat skal være et dokument som består av oppgaver samt deres løsninger i form av programkoder og algoritmer. Dette dokumentet skal være en del av databasen til et annet system.



Figur B.1: ProMat i brukermiljøet

4.2 Forutsetninger og avhengigheter

Avhengig av fremdrift i prosjektet, vil dette dokumentet endres senere. Mulige fremtidige endringer kan være for eksempel at prosjektet jobber med å finne matematisk modellering, algoritmer og programkoder for utvalgte oppgaver i matematikkfaget på både ungdomsskole og videregående skole; i slike tilfeller vil dette dokumentet oppdateres, og godkjennes av oppdragsgiveren/veilederen. Alle nye tanker og endringer skal diskuteres først og fremst med oppdragsgiveren.

5 Produktets funksjonelle egenskaper

Prosjektet er et forskningspreget prosjekt, og som forskningsmetode skal det brukes Design Science Research-metode. Metoden har tre faser:

- Relevans
- Design
- Rigor

Dette prosjektet skal sette søkelys på Rigor-fasen dvs. på forskningsdelen. Her derfor defineres ikke noen systemkrav, men defineres følgende krav til dokumentet:

- Oppgaver må velges slik at de kan bidra med dybdelæring.
- Oppgaver må velges slik at de kan stimulere engasjement og kreativitet hos elever.
- Oppgaver må lages slik at de fremviser mulige måter for algoritmisk tenkning.
- Hver oppgave må inkludere i hvert fall en instruksjon som viser hvordan oppgaven kan løses (fremgangsmåten).
- Oppgaver må ha løsningsforslag i form av enten pseudokode, programkode eller begge deler.
- Løsningsforslagene bør gi output.
- Oppgavene skal være programmerbar.

6 Ikke-funksjonelle egenskaper og andre krav

I slutten av prosjektet skal vi ha en fullstendig rapport av alt som er utarbeidet gjennom prosjektet. Det er ikke et krav at studenten lager prototype for programmerings delen.

Sara Mohammadi

Anvendelse av programmering i matematikk

NTNU
Kunnskap for en bedre verden