Computational thinking i matematik – vækstegenskaber

Frode Peulicke, Gefion Gymnasium

Denne artikel er én af fem artikler i dette nummer, der behandler emnet Computational Thinking (CT) i gymnasiefag. Denne artikel viser et eksempel på, hvordan CT kan inddrages i matematikfaget, sådan at eleverne får en dybere forståelse i faget, en forståelse som er svær at opnå på anden måde, samtidigt med at de lærer noget om programmering.

I starten af 1g, typisk efter valg af studieretning, skal eleverne forholde sig til absolut og relativ vækst. En vækst implicerer en tidsafhængighed, men i matematikundervisningen viser vi den aldrig. Normalt ser eleven bare en færdig graf, ikke hvordan den udvikler sig, når man tegner den fra et startpunkt. Eksemplet i denne tekst viser, hvordan man ser væksten fremkomme, mens tiden går. Det er min oplevelse, at elevernes forståelse af vækstypperne kan styrkes ved at lade eleverne arbejde direkte med koden i computermodellen, som illustreret i eksemplet nedenfor. Ved at arbejde med koden bliver det klart for dem, præcis hvad det er, der giver den enkelte type af vækst…

… og som bonus får de basalt kodekendskab og lærer, hvordan kode kan styre en models egenskaber.

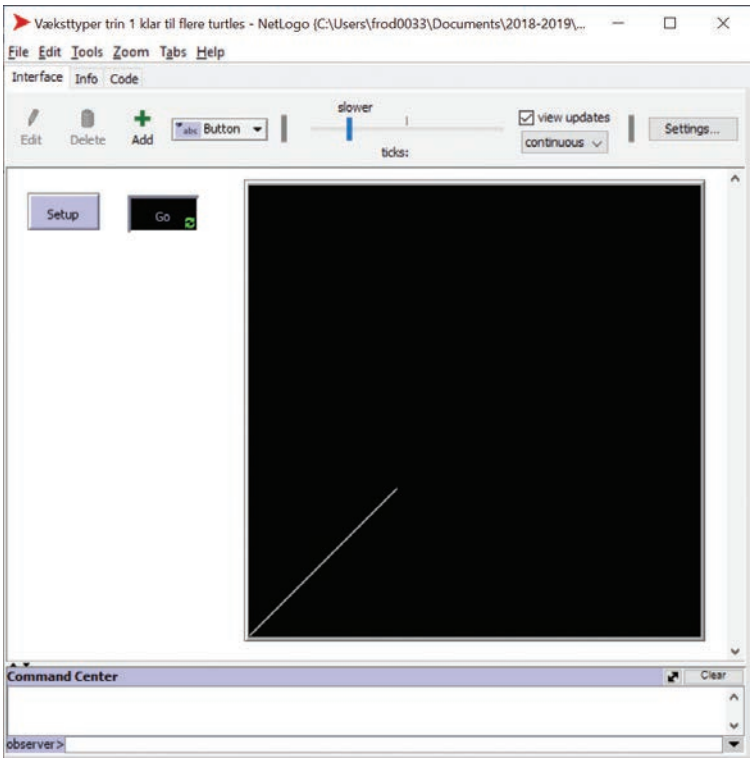
Eksemplet her i artiklen kan afvikles på cirka 45 minutter.

Undervisningsaktiviteterne

Eleverne får udleveret en NetLogo–fil, som de åbner i programmeringsmiljøet NetLogo (se side 11). Figur 1 viser det interface, som eleverne møder, når NetLogo–filen åbnes. Når der trykkes på Setup og Go, ses en ret linje kravle langsomt frem over skærmen.

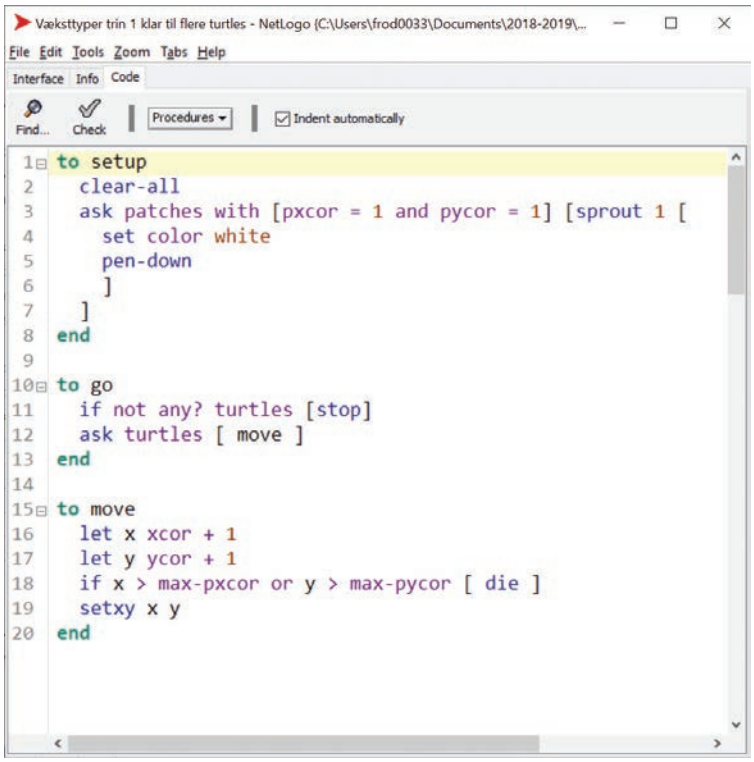
Efter at have afprøvet knapperne og erfaret hvad programmet gør, vælger eleverne fanebladet *Code*. Her ser de den computerkode, der styrer modellens opførsel, se Figur 2.

I den første opgave skal eleverne finde linjen i koden, hvor hældningen af gra-



Figur 1
Computermodellen i NetLogo viser en ret linje med hældningen 1, der er ved at blive tegnet fra nederste venstre hjørne til øverste højre hjørne.

Figur 2
Computermodellens kode. Proceduren **go** *køres i ring, når man trykker på knappen "Go" i fanebladet Interface.*



fen er bestemt og ændre den fra 1 til 2. Det resulterer i glade grynt og tilfredse miner, når det lykkes, og det er ikke bare nemt, for der er to mulige linjer, med hver sit krav til ændring. De fleste elever retter linjen, hvor *y* tælles op. Nok fordi de kender standardformuleringen "når man går 1 ud ad *x*–aksen, går man hældningstallet op ad *y*–aksen", som er meget eksplicit programmeret i algoritmen.

Bagefter bliver de bedt om at ændre d*x* (dvs. 1–tallet i linjen der starter med **let x**) i modellen og forklare, hvorfor dette også ændrer på linjens hældning.

Disse to aktiviteter kan eleverne fint lave uden forudgående kendskab til programmering eller kodning. NetLogo er lavet, så det er næsten lige så nemt at læse og forstå koden, som det er at læse og forstå engelsk. Alligevel ligner NetLogo andre programmeringssprog tilpas meget til, at dét eleverne lærer (fx mht. programstrukturer, kodesyntaks og algoritmer) kan overføres til andre programmeringssprog.

I den næste opgave skal eleverne undersøge relativ (dvs. eksponentiel) vækst. Eleverne får til opgave at ændre linjen, hvor hældningen er bestemt, så grafen bliver en eksponentiel vækst (i arbejdsarket foreslås en fremskrivningsfaktor på 1,015, men eleverne opfordres til at prøve sig frem med forskellige værdier og se, hvordan det påvirker resultatet, når modellen køres).

Denne aktivitet kræver, at eleverne tænker over, hvad eksponentiel vækst er, og hvordan *y*–koordinaten skal ændre sig, når *x*–koordinaten ændrer sig. Deres faglige viden skal oversættes til kode og sammenholdes med den resulterende kurve. Dette giver tilsyneladende eleverne en dybere forståelse af fænomenet.

For at løse opgaven skal linje 17 af koden laves om fra **let y ycor + 1** til **let y ycor * 1.015**.

Modellens interface og kode

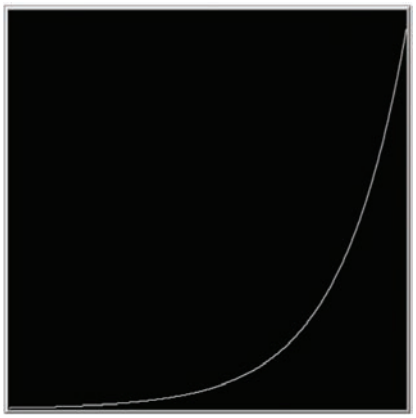
Det sorte vindue på fanebladet *Interface* er skærmen, hvor alt tegnes.

Skærmen er opbygget af *patches*, som svarer til billedpunkter (pixels). Ved at højreklikke på skærmen, kan man indstille antallet og størrelsen af disse, så skærmen får tilpas mange patches til det problem, man gerne vil arbejde med. I eksemplet har skærmen 400 patches på hver led.

Ved at anvende kommandoen **sprout**, kan der dannes en *turtle* på en bestemt patch på skærmen. En turtle svarer til en gammeldags *sprite* (for dem, der kan huske Commodore 64). Den har en form, en farve, en størrelse og et koordinatsæt. En turtle kan bevæge sig rundt på skærmen. Det kan gøres på flere måder, blandt andet ved at lave om på dens *x*– og *y*–koordinater.

Det sker i dette eksempel, hvor en turtle bevæger sig i en ret linje og trækker et spor efter sig (de patches, hvor den har været, bliver hvide).

Skærmen i interfacet ved næste kørsel ser nu ud, som vist i Figur 3.



Figur 3
En eksponentiel vækst med fremskrivningsfaktor 1,015.

Igen tilfredse udbrud blandt eleverne. Det giver en større tilfredsstillelse for eleverne at opnå disse ændringer ved reelt at ændre i koden end ved blot at ændre på en indstilling i interface. Det er i hvert fald mit indtryk, når jeg ser fistbumps og highfives i klasseværelset.

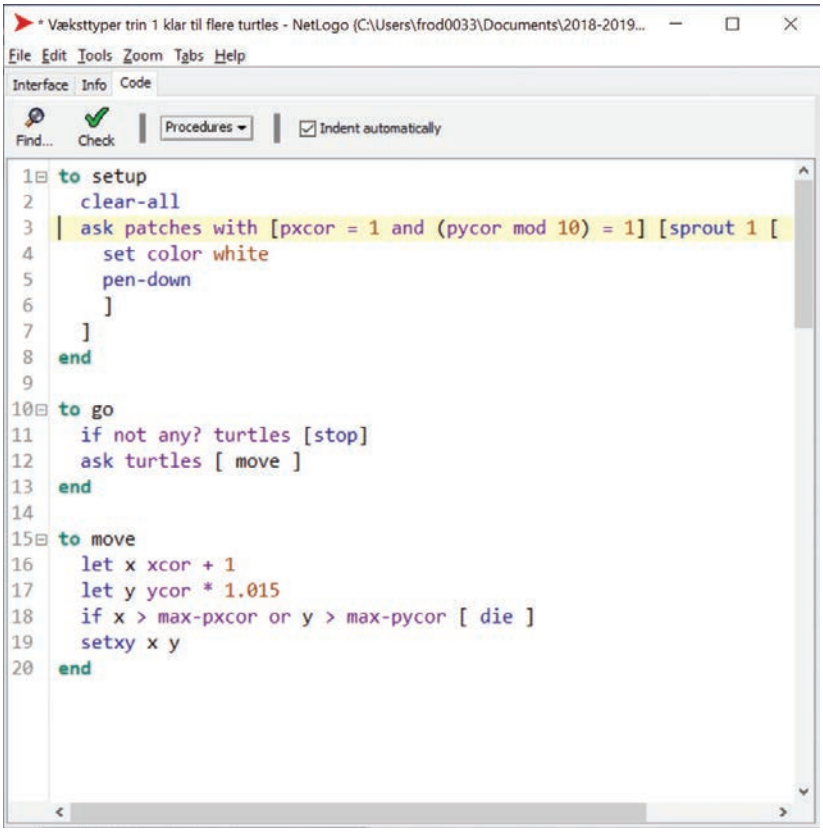
I den sidste opgave skal eleverne lave om på computermodellen, så den viser mange eksponentielle vækstkurver på samme tid (med samme fremskrivningsfaktor, men forskellige startværdier). I NetLogo, der er agentbaseret (se side 11), er det relativt nemt at omskrive koden, så der opstår mange turtles i stedet for kun én.

Da alle turtles har den samme adfærd (den relative vækst, der defineres i linje 15 – 20 af koden), har alle vækstkurverne den samme fremskrivningsfaktor. Eleverne får hjælp til at danne de mange turtles og får at vide, at de skal ændre koden fra **[pxcor = 1 and pycor = 1]** til **[pxcor = 1 and (pycor mod 10) = 1]**. Læg mærke til, at koden kun udvides med ni tegn inkl. mellemrum for at lave hele ændringen.

Koden er vist i Figur 4. Ved kørsel af programmet ser skærmen i interfacet ud, som vist i Figur 5.

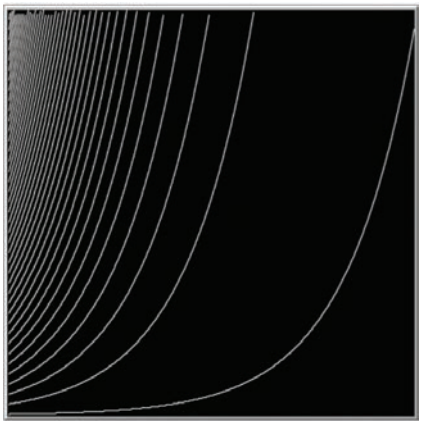
Der er stor forskel på før og efter ændringen. Eleverne spørger, hvad **mod** gør, og så får de kort præsenteret *modulo* og heltalsdivision også (på et A–niveau, ellers ville jeg nok bare sige, at den vælger hver tiende koordinat).

Kurverne har altså den samme fremskrivningsfaktor, men ser forskellige ud. Eleverne skal forklare hvorfor.



Figur 4
Koden er nu ændret, så den laver en stribe af turtles i stedet for blot en enkelt.

Figur 5
Mange eksponentielle vækstkurver med samme fremskrivningsfaktor, men forskellig startværdi.



Jeg har afprøvet dette forløb på to klasser (andre lærere har også afprøvet det på deres egne klasser) og det fører til tilfredshed og for langt de flestes vedkommende også en væsentlig dybere forståelse for begrebet fremskrivningsfaktor end 45 minutter tidligere. Klasserne kan sagtens bruge mere end 45 minutter, hvis de får lov. Så begynder de at lege med programmet og ændre farve, form og størrelse på den turtle, der tegner kurven.

Hvad fik eleverne ud af det?

Der er generelt stor tilfredshed blandt eleverne, når de oplever, at deres ændringer i koden virker og fører til de forventede ændringer, når computermodellen køres. Jeg ser det som en måde at øge elevernes motivation for faget (i det mindste i den lektion aktiviteten finder sted).

Ud over den dybere faglige forståelse oplever eleverne et break i den normale undervisning. De fleste oplever øvelsen som morsom og inspirerende.

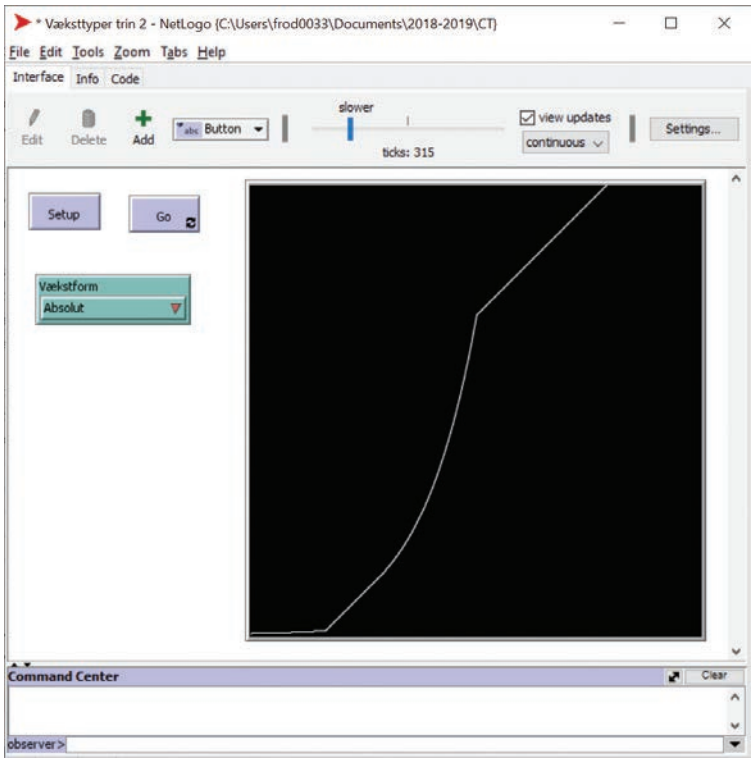
Det er en bagtanke, men for mit vedkommende ikke stillet op som succeskriterie, at eleverne også lærer lidt om programmering og udvikler CT–kompetencer undervejs. Selvom de fleste ikke direkte lærer at kode, lærer de i det mindste, at det ikke er farligt at forholde sig til og ændre i kode, de aldrig har set før. De får en oplevelse af, at dét, der står i koden, har betydning for hvordan programmet opfører sig og får et første kendskab til almindelige programstrukturer som initiering, løkker, if–sætninger og den algoritmiske opbygning af programmet (at et program består af en række konkrete instrukser til, hvad der skal ske). Som udgangspunkt får de ikke meget forklaret, men når de spørger, og det gør de, får de forklaringer til kommandoer og strukturer. Dette er et eksempel på hvordan CT kan inddrages meningsfuldt i matematikundervisningen. Jeg bruger regelmæssigt lignende CT–aktiviteter i min undervisning. Der kan findes flere forløb og undervisningsmaterialer på library.ct-denmark.org.

Materialer og videre arbejde

Materialerne og NetLogo–modellerne kan også findes på library.ct-denmark.org/lmfk. Her ligger også materialerne til et udvidet modelleringseksempel, hvor eleverne selv indfører knapper i interfacet, sådan at vækstypen kan vælges med en drop–down–menu og vækstha-stigheden og fremskrivningsfaktoren kan vælges med skydere. Det giver eleverne mulighed for at skifte mellem absolut og relativ vækst i realtid (mens programmet kører) og umiddelbart se konsekvensen, hvilket man aldrig kan gøre, hvis man kun sidder med en funktion uden nogen form for tidsfaktor.

I Figur 6 ses en kørsel med sådan et eksempel.

Et helt andet eksempel på, hvordan man kan arbejde med vækst i matematik kan findes i Eva Danielsens model med fisk library.ct-denmark.org/fang-to-punkter.



Figur 6
Skiftevis relativ og absolut vækst i samme kørsel. I det viste interface er der ikke mulighed for at vælge væksthastighed og fremskrivningsfaktor.

Forfatterens erfaringer med CT

Jeg har gennem årene flere gange prøvet at introducere CT–øvelser i fysik og matematik, men det er desværre altid faldet til jorden. Jeg har brugt masser af tid på projekter, som i sidste ende ikke førte til dybere faglig forståelse hos eleverne.

Med projektet i *Computational Thinking i Matematik og Naturfag* og udviklingsværktøjet NetLogo er det første gang, det rent faktisk er lykkedes for mig. Det er der flere grunde til, som jeg ser det.

- NetLogo–sproget er så let at forstå og så kortfattet, at selv elever helt uden programmeringserfaring kan forstå det og rette i koden (hvis koden holdes tilpas kort og letlæselig).
- Fokus ligger ikke på koden, men på fagligheden. Eleverne opnår virkelig den dybere forståelse i fagligheden, jeg har søgt tidligere, uden at de oplever, at de skal lære at programmere for programmeringens skyld. Alt hvad de laver, gør de for fagets skyld. Det kodekendskab, de får med, er ren bonus. En bonus, som de får brug for og stor glæde af i deres videre studier og i livet.
- Fordi NetLogo er agentbaseret kan det simulere rigtigt mange naturvidenskabelige fænomener på en klar og kortfattet måde.
- NetLogo er så nemt at håndtere, at jeg lige (!) kan skrive et program sammen til undervisningen… hvis jeg har en god ide. Men så har jeg også programmeringserfaring fra tidligere. Vi NetLogo–brugere gør os umage for at dele det hele og opbygge et stort bibliotek, så vi ikke skal opfinde den dybe tallerken hver gang.