Opsparingsformler

Mange problemstillinger kan betragtes som små ændringer over tid. Den mest oplagte er måske en opsparing i en sparegris. Hvis der hver måned lægges f.eks. 100 kroner i sparegrisen, vil beløbet i sparegrisen udvikle sig således:

100
$200 = 100 + 100$
$300 = 200 + 100$
$400 = 300 + 100$
osv.

For hver opsparing lægges altså blot 100 kroner til det forrige beløb. Det er ikke svært at indse, at dette kan beskrives ved formlen $100 \cdot n$, hvor n er antal måneder, der er sparet op. Lad os i stedet prøve at simulere opsparingen. Vi vil spare op i 24 måneder, og vil hver måned lægge 100 kroner i sparegrisen. Simuleringen af dette ser således ud:

```
beløb := 0 :
for i from 1 to 24 do
  beløb := beløb + 100 :
end do:
beløb
```

2400 **(1.1)**

I første linie defineres variablen *beløb*, som holder regnskab med, hvor mange penge, der er i sparegrisen. Dernæst indsættes der i 24 måneder (**from** 1 **to** 24 **do**) 100 kroner (*beløb* := *beløb* + 100). Læg især mærke til konstruktionen, som gør opsparingen 100 kroner større. Det er en central konstruktion i mange problemstillinger. Endelig printes beløbet i sparegrisen med sidste linie på skærmen.

Opgave 1

Kan du ændre simuleringen, så der i stedet spares op i 36 måneder? Og hvad nu, hvis der indsættes 150 kroner om måneden i stedet?

Simuleringen herover er mere præcist et lille program, så lad os kalde det et program i stedet. Måske skal der spares op til et bestemt beløb, og vi vil gerne undersøge, hvor lang tid, det tager. Lad os sige, at der skal spares 7000 kroner op, og at der spares 200 kroner op om måneden. I første omgang kunne vi blot prøve os frem indtil programmet når frem til det ønskede beløb. Jeg kan f.eks. af resultatet i linie **(1.1)** se, at det tager 24 måneder at spare 2400 kroner op med 100 kroner om måneden.

Opgave 2

Prøv dig frem til, hvor lang tid det tager at spare 7000 kroner op med 200 kroner om måneden.

Det ville være mere tilfredsstillende, hvis programmet blot gav tiden direkte. Det kan gøres med følgende ide: vi lader nu opsparingen køre indtil den når op over det ønskede beløb, og holder undervejs regnskab med tiden. Programmet ser således ud:

```
beløb := 0 :
tid := 0 :
while beløb < 7000 do
  beløb := beløb + 200 :
  tid := tid + 1 :
```

end do:
tid

35 **(1.2)**

Opgave 3

Læs programmet grundigt linie for linie. Forstår du, hvad der sker?

Opgave 4

Lav programmet om, så det i stedet finder tiden, det tager at spare 10000 kroner op med 350 kroner om måneden. Kontroller løsningen ved hjælp af programmet herover.

Hvis pengene i stedet blev indsat på en bankkonto, ville der (måske) blive optjent renter også. Hvis der f.eks. indsættes 100 kroner på kontoen om måneden, og der gives en rente på 1% om måneden (nok ikke en realistisk situation, men vi kan stadig lære af eksemplet), så vil beløbet på kontoen udvikle sig på følgende måde.

Måned 1	100
Måned 2	$100 \cdot 1.01 + 100$ $= 201.000$
Måned 3	$201 \cdot 1.01 + 100$ $= 303.010$
Måned 4	$303.01 \cdot 1.01$ $+ 100$ $= 406.040$
...	...

Dette kan naturligvis også simuleres. Hvis der opspares i 24 måneder, og der hver måned indsættes 100 kroner, så ser programmet til udregning af det opsparede beløb nu således ud.

beløb := 100 :
for *i* **from** 1 **to** 23 **do**
  beløb := *beløb*·1.01 + 100 :
od:
beløb

2697.346 **(1.3)**

Opgave 6

Hvorfor løber programmet kun for $1 \leq i \leq 23$? Mangler der ikke en måneds opsparing?

Opgave 7

Lav programmet om, så der i stedet opspares 350 kroner og renten er 0.1% om måneden. Opsparingen skal løbe over 36 måneder.

Opgave 8

Lad opsparingen være som i forrige opgave. Men nu skal programmet i stedet bestemme, hvor lang tid det tager, inden opsparingen er på 10000 kroner. Kontroller løsningen ved hjælp af det oprindelige program.

Hvis opsparingen skal forklares til en ikke-matematik-kyndig, så kan det være en fordel at få beløbene printet på skærmen måned for måned. Det kan gøres helt enkelt som angivet i programmet herunder (af pladshensyn opspares der kun i 5 måneder).

```
beløb := 100 :
for i from 1 to 5 do
  beløb := beløb·1.01 + 100 :
  print(beløb);
od:
```

1, 201.000

2, 303.010

3, 406.040

4, 510.101

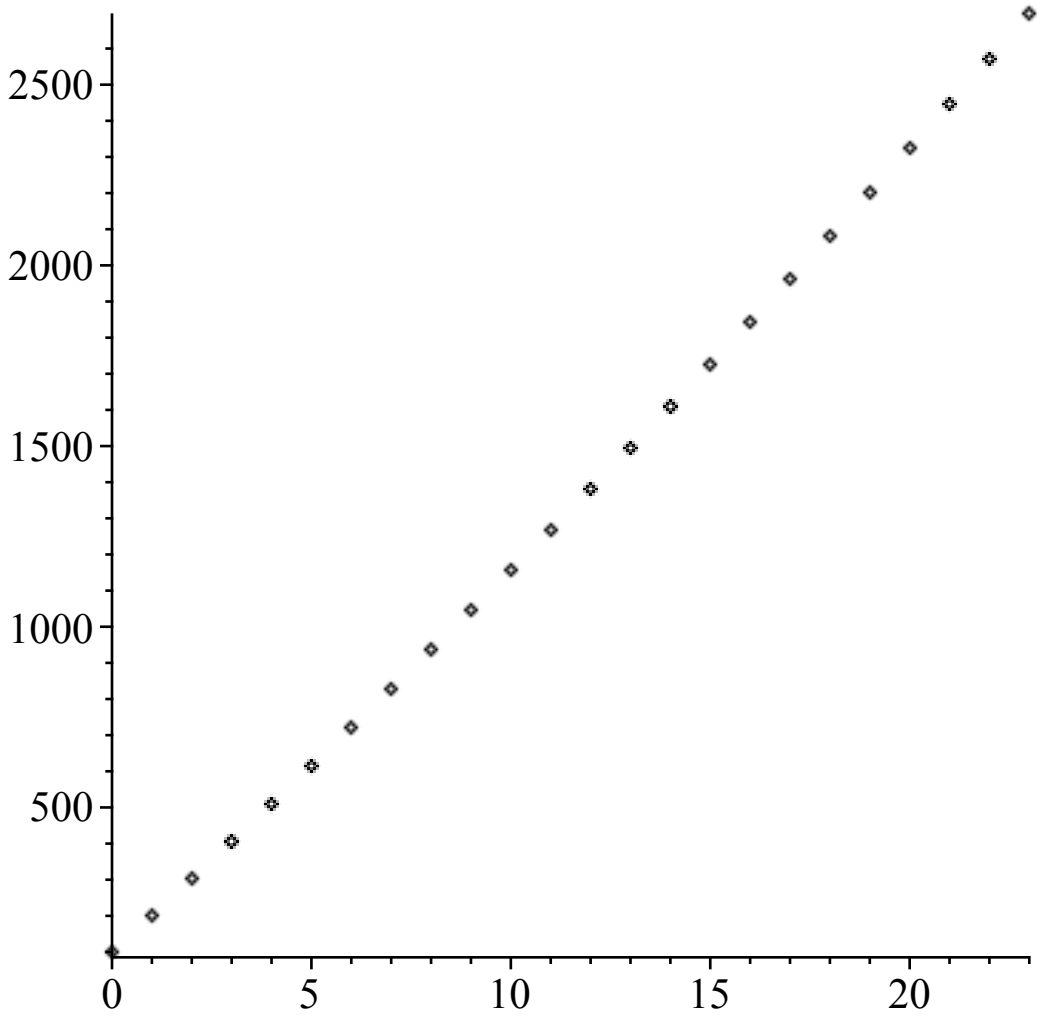
5, 615.202 **(1.4)**

Opgave 9

Lav dit program fra forrige opgave om, så det både printer det opsparede beløb måned for måned og bestemmer, hvor lang tid det tager, inden opsparingen er på 10000 kroner.
Kan du få programmet til at skrive hver måneds-nummer sammen med beløbet?

Det kan også være en fordel at præsentere opsparingen grafisk. Det kan f.eks. gøres ved at samle beløbet måned for måned på en liste, og så lave en graf over værdierne i listen bagefter. Se programmet herunder. Læg især mærke til, hvordan der defineres 2 lister (*Tider* og *Beløb*), og hvordan der skrives nye værdier i listerne (*Tider* := [*op*(*Tider*), *tid*] og *Beløb* := [*op*(*Beløb*), *beløb*]). Kommandoen *op*( ) giver alle værdierne på en liste.

```
beløb := 100 :
Beløb := [beløb] :
Tider := [0] :
for tid from 1 to 23 do
  beløb := beløb·1.01 + 100 :
  Beløb := [op(Beløb), beløb] :
  Tider := [op(Tider), tid] :
od:
with(plots) :
pointplot(Tider, Beløb);
unwith(plots)
```



Opgave 10

Lav programmet om, så det i stedet laver en grafisk præsentation af en opsparing med 220 kroner om måneden i 48 måneder med en rente på 0.4% om måneden.

2. gradspolynomier

I dette afsnit vil vi se på, hvordan et 2. gradspolynomium $f(x) = a \cdot x^2 + b \cdot x + c$ kan beskrives automatisk. I første omgang har vi brug for at aflæse koefficienterne i f, dvs. værdien af a, b og c. Dette gøres således.

$f(x) := 3\,x^2 - 7\,x + 4$:
$a := coeff(f(x), x, 2)$;
$b := coeff(f(x), x, 1)$;
$c := coeff(f(x), x, 0)$

$$a := 3$$

$$b := -7$$

$$c := 4 \qquad \textbf{(2.1)}$$

Lad os genstarte maple for at undgå at bruge disse værdier igen ved et uheld: *restart* :

Opgave 1

Lav et program, som finder diskriminant og rødder i et 2. gradspolynomium.

Opgave 2

Lav et program, som finder toppunktet for en parabel.

Programmet ovenover kan med fordel samles i en *procedure*. Vi kan tænke på en procedure som en *kommando* i maple. Proceduren skal hedde *koefficienter*, og defineres således:

```
koefficienter := proc(f)
 local a, b, c :
 a := coeff(f, x, 2) :
 b := coeff(f, x, 1) :
 c := coeff(f, x, 0) :
 return [a, b, c] :
end proc:
```

Linien
local a, b, c :
sikrer, at a, b og c kun gives værdier mens programmet kører, og derefter slettes igen. Vi bliver altså fri for at genstarte maple.

Lad os prøve proceduren (eller programmet):

```
f(x) := 3 x^2 - 7 x + 4 :
koefficienter(f(x))
```

$$[3, -7, 4] \qquad \textbf{(2.2)}$$

Vi kan også skrive polynomiet direkte som input til programmet.

```
koefficienter(3 x^2 - 7 x + 4)
```

$$[3, -7, 4] \qquad \textbf{(2.3)}$$

Hvis jeg kun er interesseret i b-værdien, kan jeg få den på følgende måde.

```
L := koefficienter(f(x)) :
L[2]
```

$$-7 \qquad \textbf{(2.4)}$$

a-værdien fås tilsvarende som $L[1]$ og c-værdien som $L[3]$.

Opgave 3

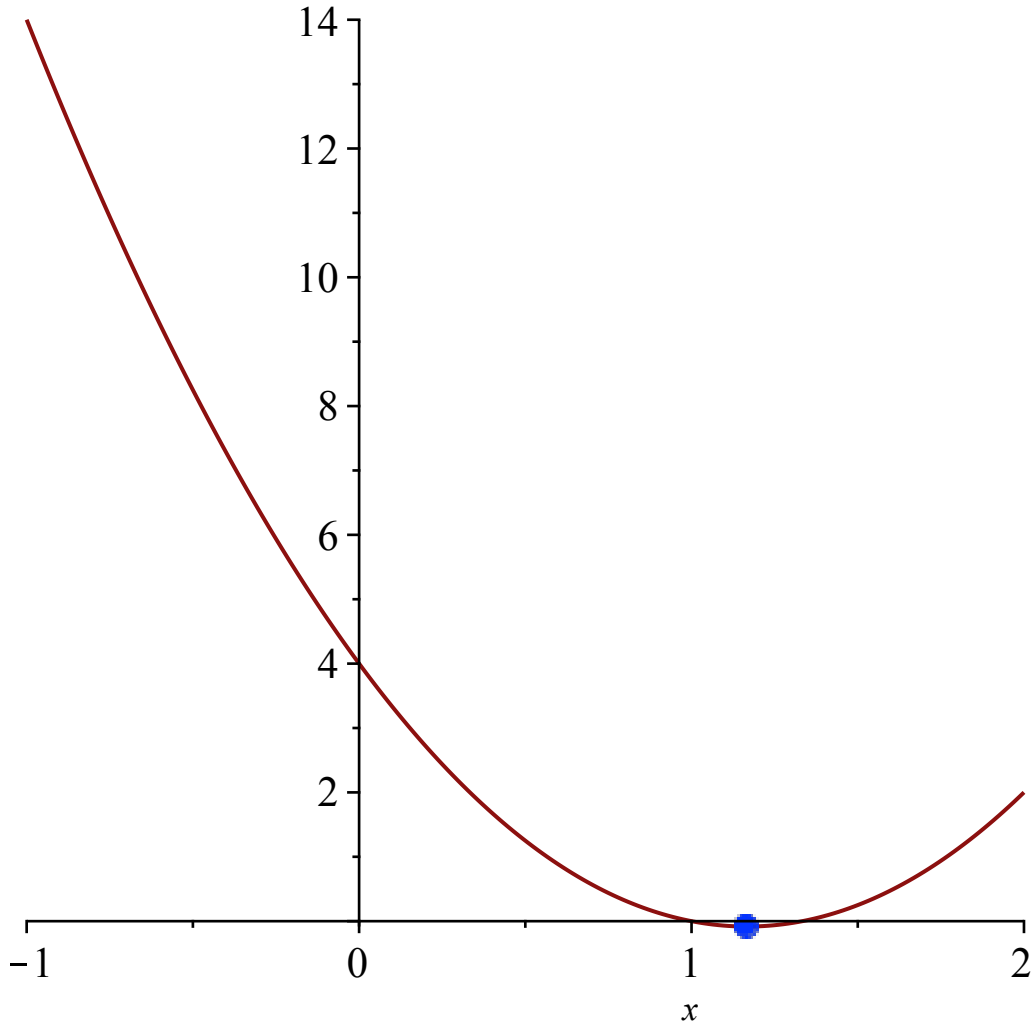
Lav en procedure, som finder rødderne i et 2. gradspolynomium.
Kan du bruge programmet *koefficienter* i din procedure? Husk at definere programmet ved at klikke på det og trykke "enter".

Opgave 4

Lav en procedure, som finder toppunktet for en parabel.

Procedurer kan f.eks. bruges til at lave flere beregninger og plots på een gang. Hvis jeg vil tegne grafen for et 2. gradspolynomium sammen med dets toppunkt (markeret med et rødt punkt), så kan jeg gøre det således:

```
with(plots):
f(x) := 3*x^2 - 7*x + 4:
Tx := solve(f'(x) = 0):
plot1 := plot(f(x), x = -1..2):
plot2 := pointplot([Tx, f(Tx)], color = blue, symbol = solidcircle, symbolsize = 18):
display([plot1, plot2])
```



Opgave 5

Lav en procedure, som tegner grafen for et 2. gradspolynomium sammen med toppunktet. Toppunktet skal være markeret med et rødt punkt, og grafen skal være sort.

Skrå kast

Konstruktion med at regne fremad med små tidsskridt dt og finde nye x- og y-koordinater. Samspil mellem simulering og integralløsning. Ende ud med at tegne graf for skråt kast på baggrund af starthastighed og retning.

Betragt en partikel, som bevæger sig med konstant hastighed. At hastigheden er konstant betyder, at strækningen x, partiklen tilbagelægger i lige store tidsintervaller, er konstant. Vi kan altså udregne, hvor partiklen er efter N tidsintervaller på følgende måde.

```
x := 0 :
Δx := 2 :
N := 10 :
for t from 1 to N do
  x := x + Δx :
od:
x
```

$$20 \qquad \textbf{(3.1)}$$

Partiklen starter her med at have bevæget sig 0 meter, og bevæger sig i hvert tidsinterval 2 meter.

Hvis partiklen bevæger sig med hastighed v, så bevæger den sig $v \cdot \Delta t$ meter i løbet at Δt sekunder. Vi kan med denne notation beregne, hvor langt partiklen kommer på f.eks. 30 sekunder med hastigheden 5 meter pr. sekund.

```
x := 0 :
t := 0 :
Δt := 0.01 :
v := 5 :
while t < 30 do
  t := t + Δt :
  x := x + v·Δt :
end do:
x
```

$$150.000 \qquad \textbf{(3.2)}$$

Opgave 1

En løber starter med at løbe 5 meter fra dig, og løber væk fra dig med hastigheden 4 meter pr. sekund. Skriv et program, som kan udregne, hvor langt væk løberen er fra dig efter 2 minutter.

Opgave 2

Du prøver nu at indhente løberen fra opgave 1. Din hastighed er 5 meter pr. sekund. Hvornår indhenter du løberen? Skriv et program, som kan løse denne opgave for dig.

Betragt nu en partikel, som bevæger sig med konstant acceleration. Det betyder, at hastigheden ændrer sig med den samme størrelse Δv i lige store tidsintervaller. Vi kan udregne, hvor partiklen er efter N tidsintervaller på følgende måde.

```
x := 0 :
t := 0 :
Δt := 0.01 :
v := 2 :
```

```
Δv := 0.005 :
while t < 120 do
  t := t + Δt :
  x := x + v·Δt :
  v := v + Δv :
od:
 x
```

3839.700 **(3.3)**

Her starter partiklen med at have bevæget sig 0 meter, og har til start hastigheden 2 meter pr. sekund. Partiklen betragtes i 2 minutter, og i programmet indbygges, at hastigheden ændrer sig med 0.005 meter pr. sekund for hver gang, der går 0.01 sekund.

Hvis partiklen acceleration er på a meter pr. sekund for hvert sekund (dvs. hastigheden ændrer sig med a meter pr. sekund for hvert sekund, der går), så ændres hastigheden sig med $a \cdot \Delta t$ i løbet at Δt sekunder. Vi kan med denne notation beregne, hvor langt partiklen kommer på f.eks. 30 sekunder med begyndelseshastigheden 5 meter pr. sekund og en acceleration på 2 meter pr. sekund pr. sekund.

```
x := 0 :
t := 0 :
Δt := 0.01 :
v := 5 :
a := 2 :
while t < 30 do
  t := t + Δt :
  x := x + v·Δt :
  v := v + a·Δt :
od:
 x
```

1049.700 **(3.4)**

Opgave 3

En løber starter med at løbe 5 meter fra dig. Hun starter med at stå stille, og accelererer med 1.2 meter pr. sekund pr. sekund væk fra dig. Skriv et program, som kan udregne, hvor langt væk hun er fra dig efter 10 sekunder.

Opgave 4

Du prøver nu at indhente løberen fra opgave 1. Din egen acceleration er 1.8 meter pr. sekund. Hvornår indhenter du løberen? Skriv et program, som kan løse denne opgave for dig.

Du kan antage, at I begge holder samme acceleration, selvom dette nok er urealistisk i en virkelig situation. Kan du evt. ændre programmet, så accelerationen bliver 0 når tophastigheden er nået? Du kan måske bruge denne programstump:

```
if v = v_max then
  a := 0 :
end if:
```

Et skråt kast består at 2 bevægelser: en vandret bevægelse med konstant hastighed, og en lodret bevægelse med konstant acceleration på $g = -9.82$ meter pr. sekund pr. sekund (negativt fortegn fordi positiv retning er opad, dvs. modsat tyngdeaccelerationen). Hvis vi antager, at der ikke er nogen tyngdekraft, så er den lodrette bevægelse også med konstant hastighed, og vi kan beregne partiklens position (x, y) på følgende måde.

```
x := 0 :
y := 0 :
t := 0 :
Δt := 0.01 :
vx := 3 :
vy := 2 :
while t < 30 do
  t := t + Δt :
  x := x + vx·Δt :
  y := y + vy·Δt :
end do:
[x, y]
```

$$[90.000, 60.000] \qquad \textbf{(3.5)}$$

Opgave 5

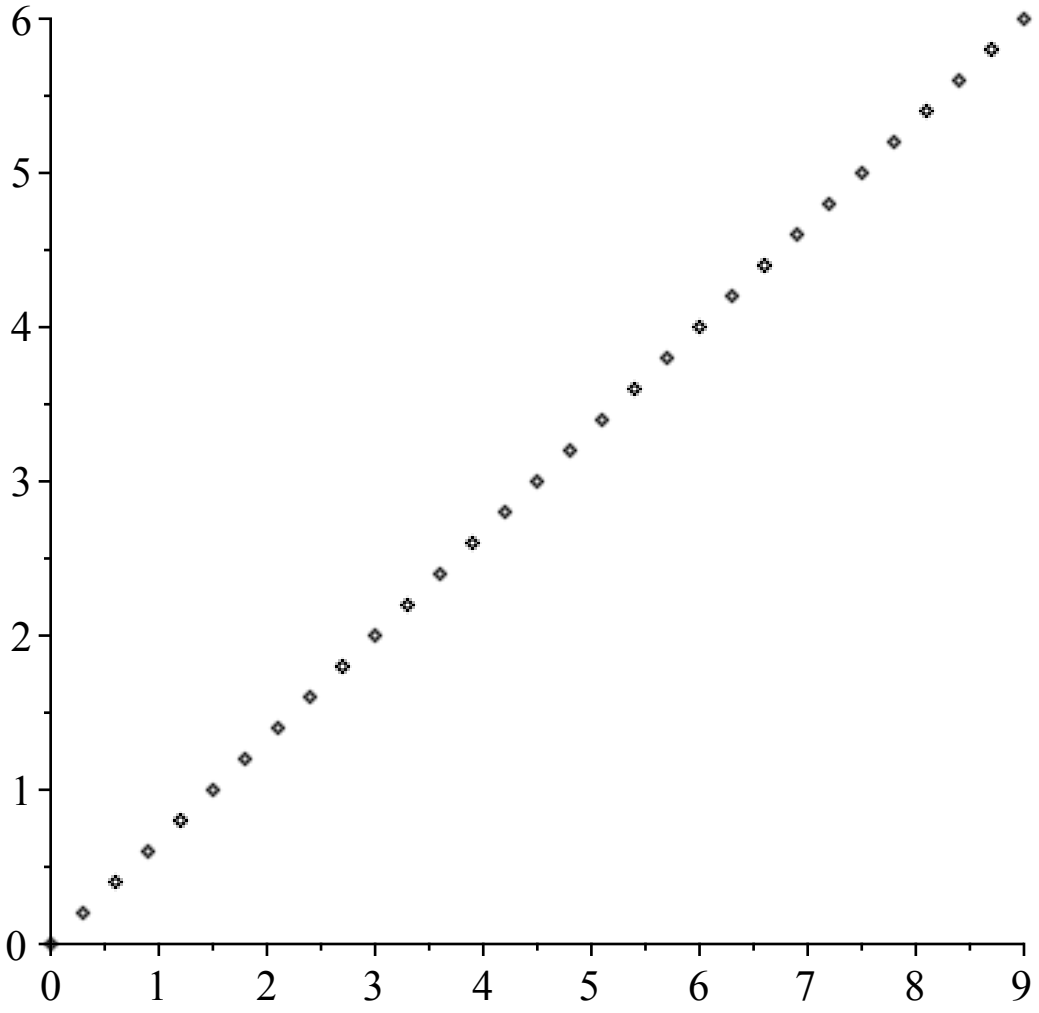
Lav programmet om, så bevægelsen i y-retningen er med den konstante acceleration -9.82 meter pr. sekund pr. sekund.

Hvis vi vil følge partiklen undervejs, så skal vi blot opdatere lister undervejs, som holder regnskab med partiklens x- og y-værdier. Herunder antages igen, at partiklen ikke accelereres i lodret retning.

```
x := 0 :
y := 0 :
xListe := [x] :
yListe := [y] :
t := 0 :
Δt := 0.1 :
vx := 3 :
vy := 2 :
while t < 3 do
  t := t + Δt :
  x := x + vx·Δt :
  y := y + vy·Δt :
  xListe := [op(xListe), x] :
  yListe := [op(yListe), y] :
end do:
```

Med et punkt-plot kan vi nu se bevægelsen illustreret.

```
plots[pointplot](xListe, yListe)
```

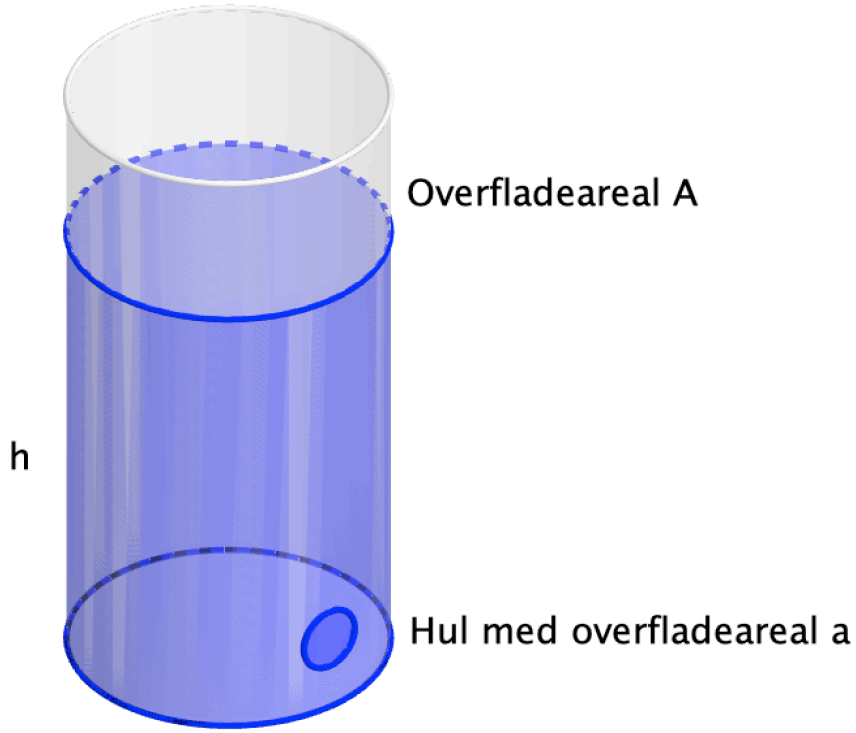


Opgave 6

Lav dit program fra forrige opgave om, så du som herover kan se bevægelsen punkt for punkt i stedet for blot sidste punkt i bevægelsen. Kan du se, hvilken type kurve, punkterne ligger på?

Vandbassiner

Betragt følgende model af et vandbassin. Den blå del skal forestille en væskesøjle med højden h.



Vandet løber ud af spanden pga. tyngdekraften. Fra fysik fås, at vandets hastighed v ud af spanden er giver ved $v = \sqrt{2 \cdot g \cdot h}$, hvor $g = 9.82 \frac{\text{m}}{\text{s}^2}$. Hvis vandet ikke komprimeres, er $-A \cdot \Delta h = a \cdot v \cdot \Delta t$ (negativt fordi h bliver mindre, dvs. $\Delta h \leq 0$; hvad udtrykker ligningen?). Derfor er $-A \cdot \Delta h = a \cdot \sqrt{2 \cdot g \cdot h} \cdot \Delta t$, hvorfor $\Delta h = \frac{a \cdot \sqrt{2 \cdot g \cdot h} \cdot \Delta t}{-A}$.

Du skal nu lave et program (en model!), som simulerer vandstanden i spanden. Måske er det en fordel at lave programmet trinvist.

1. Lav en simpel model, hvor der løber vand med konstant hastighed ned i en lukket spand (en badekars-model).
2. Lav et program, som simulerer situationen herover, hvor vandet løbet ud af spanden, og spanden er fyldt til at starte med.
3. Tag højde for, at programmet skal stoppe når spanden er tom.
4. Udvid programmet, så der også er et tilløb til spanden, og lad tilløbet være med konstant hastighed.
5. Ændr programmet, så tilløbet i stedet er fra en fyldt spand med hul i (dvs. 2 spande, begge med hul i).
6. Udvid programmet, så der er tilløb til den 1. spand.
7. Tilføj mulighed for at have forskellige størrelser huller i spandene.
8. Lav plot at vandstanden i de 2 spande.
9. Udforsk din model i forhold til en virkelig situation med vandbasssiner a'la dem, Skanderborg Forsyning bruger til vandrensning.