Norges teknisk–naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

TDT4105 IT Grunnkurs
Høst 2014

Øving 6

1 Teori

a) Hva er 2-komplement?

b) Hva er en *sample* innen digital representasjon av lyd?

c) Hvorfor kjører skript nummer 2 tregere enn nummer 1?

```
x = zeros(10000)
for i = 1:10000
  x(i) = i;
end
```

```
for i = 1:10000
  x(i) = i;
end
```

2 Kodeforståelse

Hva skriver følgende kode ut? Gå gjennom koden først, og noter deg hva du mener den skriver ut, før du kjører den.

```
function c = aFunction(a, b)
  c = a + b;
  b = 3;
end
```

```
b = 5;
c = 2;
d = aFunction (c, b);

disp(b);
disp(c);
disp(d);
```

3 Trapesmetoden

a) Lag funksjonen `f`, gitt av:

$$f(x) = 8sin(x)^x - 5$$

Pass på at funksjonen kan regne på vektorer.

Tips: a ^ b fungerer kun hvis a og b er tall, a .^ b fungerer også hvis a og b er vektorer.

Følgende skript tester funksjonen:

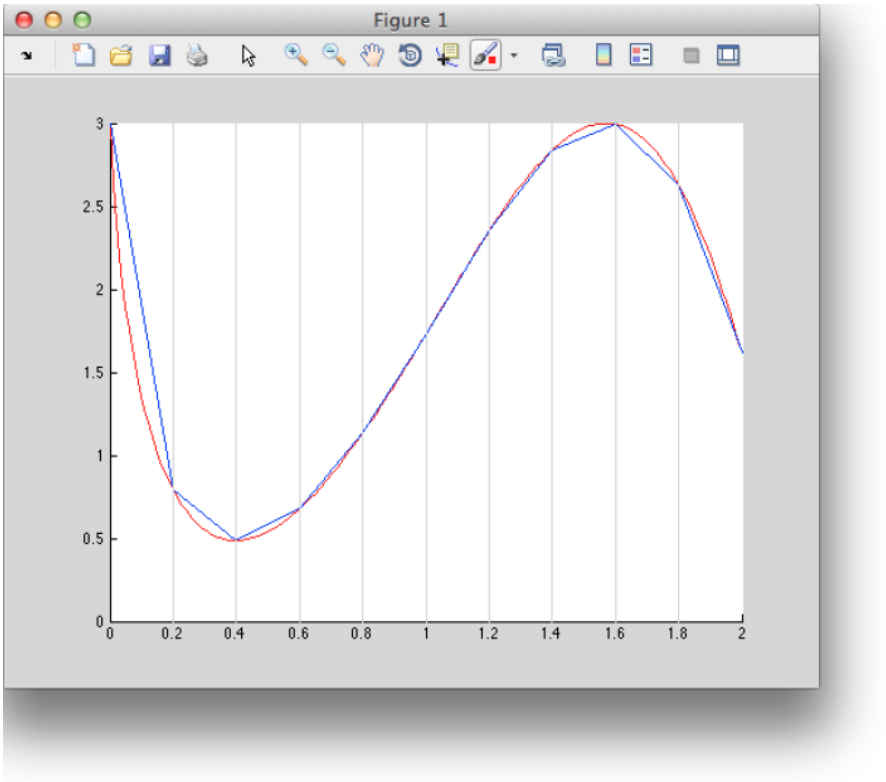
```
f(0) % skal returnere 3
f(1.5) % skal returnere 2.9700

% skal returnere [3.0000 0.5392 1.7318 2.9700 1.6146]
f([0:0.5:2])
```

b) Plott funksjonen fra 0 til 2 med steglengde 0.01.

c) Plott funksjonen fra 0 til 2 med steglengde 0.2

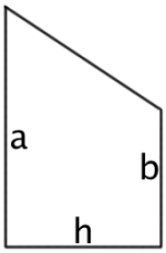
Hvis du skriver `hold on` før du kjører `plot`-funksjonen, vil plottene legge seg oppå hverandre.

Hvis du har gjort dette vil plottet se ca. slik ut:



For å finne arealet under en graf er det vanlig å integrere funksjonen. Men det er ikke alle funksjoner som er enkle å integrere. Funksjonen i denne oppgaven er et eksempel på en slik funksjon. Men med litt programmering kan vi tilnærme oss arealet av denne funksjonen.

Se på plottet ovenfor, og legg merke til at de to grafene nesten ligger oppå hverandre. Legg også merke til at den blå linjen, sammen med bunnlinjen og de vertikale grå linjene, danner 10 trapeser. Trapes er det enkelt å regne ut arealet av.



d) Lag funksjonen `trapezoidArea` som tar inn parametrene a, b og h. Bruk følgende funksjon for å regne ut arealet.

$$A = \frac{a+b}{2} * h$$

Her er a og b lengden på de to parallelle sidene i trapeset, og h er distansen mellom disse.

Test funksjonen slik:

```
trapezoidArea(1, 1, 1) % skal returnere 1
trapezoidArea(3, 1, 1) % skal returnere 2
trapezoidArea(1, 3, 2) % skal returnere 4
```

e) Lag funksjonen `trapezoidMethod`, som skal regne ut en tilnærming av arealet under funksjonen `f` fra oppgave a). Funksjonen skal ta inn `start`, `stop` og `n` som parametre. Her er `start` start-punktet for arealet (0 i eksempelet over), `stop` er stopp-punktet (2 i eksempelet over) og `n` er antall trapeser.

Test funksjonen slik:

```
trapezoidMethod(0, 2, 10) % skal returnere 3.5875
trapezoidMethod(0, 2, 2) % skal returnere 4.0391
trapezoidMethod(1, 3, 10) % skal returnere 0.1475
```

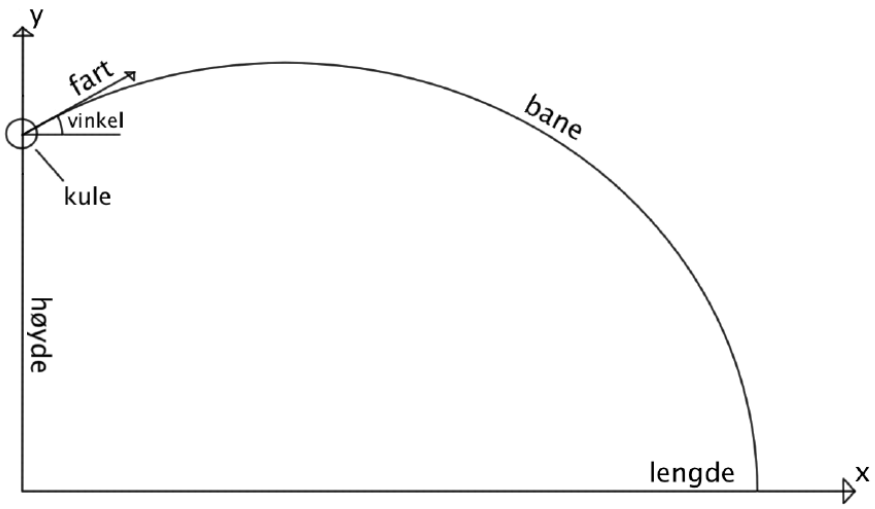
Hva skjer når du øker n?

4 Kulestøt

Når man støter kule er det ikke bare hastigheten på kulen som avgjør hvor langt man støter - vinkelen man støter med og høyden over bakken spiller også en rolle. I denne oppgaven skal vi lage en funksjon som modellerer dette problemet, og vi skal finne ut hvilken vinkel som er optimal for en gitt høyde.

Vi skal regne på bevegelse i to dimensjoner, `x` og `y`. Det er ikke nødvendig å forstå fysikken bak bevegelsene, siden du skal få oppgitt riktige formler og hvor de skal brukes. Men vi må likevel ha en liten intuisjon for hva som foregår.

Hvis høyden er lik 2 så er startposisjonen x = 0 og y = 2.



For å kalkulere banen som kulen tar, trenger vi å vite hvor kulen er, hva farten er og hvilken akselerasjon den har. Videre lar vi det gå en tidsenhet, slik at kulen flytter seg litt og kalkulerer disse variablene på nytt. Dette gjentar vi så lenge y-koordinatet er større enn 0. Når y-koordinatet er 0 har kulen landet, og lengden av kastet er gitt av x-koordinatet.

Vi vet allerede hvor kulen er når den starter, så det første vi må kalkulere er hastigheten i henholdsvis x-retning og y-retning. Dette er gitt av hastigheten og vinkelen ved følgende formler:

$$v_x = cos(angle) * speed$$

$$v_y = sin(angle) * speed$$

a) Skriv funksjonen `initialSpeed(angle, speed)` som returnerer hastighet i `x` og `y` retning.
Tips: cos og sin i matlab regner med radianer og ikke grader. Hvis du vil bruke grader i løsningen din kan du bruke funksjonene cosd og sind. Eller du kan gjøre omregningen selv. Husk at $grader = radianer * \frac{180^\circ}{\pi}$
Funksjonen kan testes med følgende skript:

```
% skal returnere x = 100, y = 0
[x, y] = initialSpeed(0, 100)
% skal returnere x = 6.1232e-15, y = 100
[x, y] = initialSpeed(pi/2, 100)
% skal returnere x = 70.7107, y = 70.7107
[x, y] = initialSpeed(pi/4, 100)
```

Posisjonen i neste steg forholder seg til posisjonen og farten i det nåværende steget slik:

$$x_{n+1} = x_n + v_{xn} * dt$$

$$y_{n+1} = y_n + v_{y_n} * dt$$

`dt` er her størrelsen på et tidsintervall.

b) Skriv funksjonen med signaturen `function [x, y] = position(x, y, vx, vy, dt)` som kalkulerer x- og y-koordinatet i neste steg, ut ifra det nåværende stegets posisjon og fart.
Funksjonen kan testes med følgende skript:

```
% skal returnere x = 10, y = 11
[x, y] = position(10, 10, 0, 1, 1)
% skal returnere x = 11, y = 10
```

```
[x, y] = position(10, 10, 1, 0, 1)
% skal returnere x = 10.1, y = 10.1
[x, y] = position(10, 10, 1, 1, 0.1)
```

Akselerasjonen er i denne oppgaven gitt ved:

$$a_{xn+1} = -k * v_{xn} * abs(v_{xn})$$
$$a_{y_{n+1}} = -k * v_{y_n} * abs(v_{y_n}) - g$$

Hvor `k` = 0.01 og angir luftmotstanden, og `g` er gravitasjonskonstanten (9.81 på jorden).

c) Skriv en funksjon med signaturen `function [ax, ay] = acceleration(vx, vy)` som kalkulerer akselerasjonen i det neste steget ut i fra det nåværende stegets fart.

Funksjonen kan testes slik:

```
% skal returnere x = 0, y = -9.81
[x, y] = acceleration (0, 0)
% skal returnere x = -1, y = -10.81
[x, y] = acceleration (10, 10)
```

Farten i neste steg forholder seg til farten og akselerasjonen i det nåværende steget slik:

$$v_{xn+1} = v_{xn} + a_{x_n} * dt$$
$$v_{y_{n+1}} = v_{y_n} + a_{y_n} * dt$$

`dt` er her størrelsen på et tidsintervall.

d) Skriv en funksjon med signaturen `function [vx, vy] = speed(vx, vy, ax, ay, dt)` som kalkulerer farten i det neste steget ut ifra det nåværende stegets fart og akselerasjon.

Nå kan vi bruke funksjonen i a for å finne starttilstanden og funksjonene i b-d for å gå fra et steg til det neste.

e) Skriv funksjonen `trajectory` som tar inn vinkel, fart og høyde over bakken som parameter. Funksjonen skal returnere to vektorer, en for alle x-koordinatene og en for alle y-koordinatene. Siden vi ikke vet hvor mange iterasjoner løkken skal inneholde, må funksjonen benytte en `while`-løkke. Løkken skal stoppe når y-koordinatet er mindre enn 0.

f) Lag funksjonen `plotTrajectory` som har de samme parameterene som `trajectory`, men som bruker `plot`-funksjonen til å plotte kulebanen.

g) (frivillig) Lag funksjonen `plotTrajectoryLength`. Den skal kalle `trajectory` med forskjellige vinkler mellom 0 og pi/2 (90°), for å finne lengden av et kulestøt med den aktuelle vinkelen. Deretter skal funksjonen plotte vinkel mot lengde ved hjelp av `plot`.

Et greit vinkelsteg her kan være pi / 32.

Hvilken vinkel gir best resultat?

Hva skjer om man endrer på høyden; endrer vinkelen seg da?

Hva skjer om man endrer på k og g?

Har farten noe å si for vinkelen?

Klarer du å lage en 3D plot med høyde og vinkel?