Programmering og Modellering (PoM)
Ugeseddel 14 — Uge 51 — Deadline 9/1

Kim Steenstrup Pedersen, Katrine Hommelhoff Jensen, Knud Henriksen,
Mossa Merhi og Hans Jacob T. Stephensen

15. december 2014

1 Plan for ugen

Denne uge tager vi fat på et nyt emne: Modellering med partikelsystemer. En partikelsimulator, også betegnet en *N-legeme simulator* er et system der simulerer en større mængde partiklers adfærd, under indflydelse af forskellige fysiske kræfter. Partikelsimulatorer bliver typisk anvendt til at studere komplekse systemer i fysik og astronomi, fra planetsystemer, stjerneklynger og dannelsen af galakser til kemiske processer såsom vandmolekuler, gasmolekuler og ild. Fælles for disse systemer er at partiklerne indgår som selvstændige objekter der reagerer på udefrakommende kræfter, som f.eks. kan være tyngdekraften, vind, temperatur, kollisioner med og tiltrækning/frastødning af andre partikler. Til de store, beregningstunge systemer kræves en effektiv implementation og typisk parallelberegninger på en klynge af processorer. I skal nu stifte bekendtskab med en simpel partikelsimulator, hvor I får mulighed for selv at vælge et passende design.

Til tirsdag:
Læs: Ikke noget. Men tænk rigtig meget på kode :)
Til forelæsningen gennemgås:

- N-legeme problemet
- Objektorienteret designtilgang
- Lidt repetition af vektoraritmetik

Til torsdag:
Læs: Som tirsdag.
Til forelæsningen fortsættes fra tirsdag.

1.1 Gruppeopgave

Den 9/1 senest klokken 15:00 skal besvarelse af følgende opgave afleveres elektronisk via Absalon. Opgaven skal besvares i grupper bestående af 1 til 3 personer og skal godkendes, for at gruppedeltagerne kan kvalificere sig til den afsluttende tag-hjem eksamen. Opgavebesvarelsen skal oploades via kursushjemmesiden på Absalon (find underpunktet `ugeseddel14` under punktet `Ugesedler og opgaver`). Kildekodefiler ("script"-filer) skal afleveres som "ren tekst", sådan som den dannes af `emacs`, `gedit`, `Notepad`, `TextEdit` eller hvilket redigeringsprogram man nu bruger (*ikke* PDF eller HTML eller RTF eller MS Word-format). Filen skal navngives *efternavn1.efternavn2.efternavn3*.51.py, mens andre filer skal afleveres som en PDF-fil med navnet *efternavn1.efternavn2.efternavn3*.51.pdf.

14g1 Denne opgave handler om at simulere gasmolekuler i en todimensionel idealgas.

2d vektoroperationer

I ugens opgave skal vi blandt andet arbejde med vektoraritmetik. En vektor betegner vi her med fed skrift, og er givet ved tuplen $\mathbf{v} = (v_x, v_y)$. Vi vil anvende den samme notation for punkter. Nedenfor er beskrevet en række aritmetiske operationer, I får brug for at implementere i opgaven. (Hint: Man kunne for eksempel anvende `numpy.mat`).

- **Længden $|\mathbf{v}|$ af en vektor $\mathbf{v}$** fås ved

$$\text{len}(\mathbf{v}) = \sqrt{v_x^2 + v_y^2} \tag{1}$$

- Skalering af vektoren $\mathbf{v}$ med en **længde specificeret af skalarværdien** s fås ved

$$\text{scale}(\mathbf{v}, s) = \left(\frac{v_x}{|\mathbf{v}|} s, \frac{v_y}{|\mathbf{v}|} s \right) \tag{2}$$

- **Vektoren udspændt mellem to punkter $\mathbf{p1} = (p1_x, p1_y)$ og $\mathbf{p2} = (p2_x, p2_y)$** (fra **p1** til **p2**) fås ved

$$\text{vec}(\mathbf{p1}, \mathbf{p2}) = (p2_x - p1_x, p2_y - p1_y) \tag{3}$$

- **Additionen af to vektorer $\mathbf{v1} = (v1_x, v1_y)$ og $\mathbf{v2} = (v2_x, v2_y)$** fås ved

$$\text{add}(\mathbf{v1}, \mathbf{v2}) = (v1_x + v2_x, v1_y + v2_y) \tag{4}$$

- **Prikproduktet** mellem to vektorer $\mathbf{v1} = (v1_x, v1_y)$ og $\mathbf{v2} = (v2_x, v2_y)$ fås ved

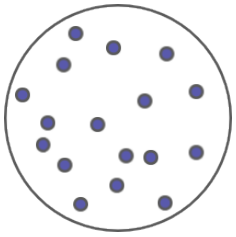
$$\text{dot}(\mathbf{v1}, \mathbf{v2}) = v1_x v2_x + v1_y v2_y \tag{5}$$

- **Projektionen af et punkt $\mathbf{p}$ på en vektor $\mathbf{v}$** fås ved

$$\text{proj}(\mathbf{p}, \mathbf{v}) = \frac{\text{dot}(\mathbf{p}, \mathbf{v})}{|\mathbf{v}|} \frac{\mathbf{v}}{|\mathbf{v}|} \tag{6}$$

Opgave: 2d gasmolekulesimulator

Vi vil betragte molekulerne som punktformede partikler med en masse og vi anvender en simplificeret model for en idealgas [1]. Gaspartiklerne er indeholdt i en rund beholder, hvor de uafhængigt af hinanden bevæger sig med en konstant hastighed og kun ændrer retning ved kollision med beholderens væg:



Systemet betragtes i tidsintervallet $[0, t_{max}]$, hvor $0 \leq t_i \leq t_{max}$ betegner de enkelte tidsskridt, således at størrelsen af et tidsskridt er Δt. Simulatoren vil indeholde to typer af objekter:

[1] `http://en.wikipedia.org/wiki/Kinetic_theory`

- **Partikel**: Dette objekt er beskrevet ved en positionsvektor $\mathbf{p} = (p_x, p_y)$, en hastighedsvektor $\mathbf{v} = (v_x, v_y)$, hvor længden af $\mathbf{v}$ bestemmer farten som partiklen bevæger sig med og retningen af bevægelsen er givet ved vektoren. Derudover har partiklen en masse, som vi dog i det efterfølgende vil antage er $m = 10^{-23}$ g (gram). En partikel initialiseres med en bestemt position og hastighed inde i beholderen. Efter initialiseringen opdateres $\mathbf{p}$ ved hvert tidsskridt Δt som

$$\text{step}(\mathbf{p}, \Delta t) = (p_x + \Delta t v_x, p_y + \Delta t v_y) \tag{7}$$

- **Beholder**: Dette objekt er beskrevet ved en radius r, en centrumsposition $\mathbf{c} = (c_x, c_y)$ samt en liste af partikelobjekter. En beholder initialiseres med en radius og eventuelt en centrumsposition, som alternativt kan sættes til $(0, 0)$. Den skal selv oprette listen med partikelobjekterne, enten i initialiseringen eller v.h.a. en funktion, begge med en parameter som angiver antallet af partikler. Beholderen skal sørge for at initialisere hver partikel med en tilfældig position, men dog sikre at positionen er indenfor beholderens væg. Derudover skal hastighedsvektorene også initialiseres med tilfældige værdier og her foreslår vi at man sampler en retning ved at trække en uniformt fordelt vinkel fra intervallet $\theta \in [0; 2\pi]$ og at man sampler farten (dvs. længden af hastighedsvektoren) uniformt fra et interval $v \in [0; v_{\max}]$. Vi kan konvertere disse to tilfældige værdier til en hastighedsvektor ved at anvende følgende formel

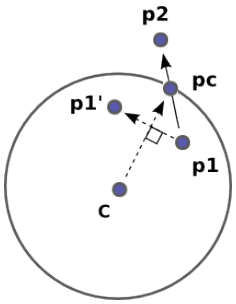
$$\mathbf{v} = (v \cdot \cos\theta, v \cdot \sin\theta) \ .$$

  vi har brug for en øvre grænse på farten $v_{\max}$ som bør sættes således at partiklen skal bruge mange tidsskridt for at bevæger sig fra den ene side af beholderen til den anden.

Der skal i hvert tidsskridt undersøges for kollisioner af partikler med beholderens væg. En simpel måde at gøre dette er, i forbindelse med opdateringen af en partikels nye position via (7), at tjekke om denne position vil føre partiklen udover beholderens rand, dvs. om afstanden til beholderens centrum er større end dens radius. Dette kan gøres med følgende boolske operation

$$\text{willcollide}(\mathbf{p}, \mathbf{v}, \mathbf{c}, r) = \text{len}(\text{vec}(\mathbf{c}, \text{step}(\mathbf{p}, \Delta t))) > r \tag{8}$$

hvor $\mathbf{p}$ og $\mathbf{v}$ er partiklens nuværende position og hastighed, $\mathbf{c}$ beholderens centrum og r beholderens radius.



Figur 1: Kollisionssituation mellem partikel og beholderens væg: **p1** er partiklens nuværende position, **p2** = step(**p1**, Δt) dens potentielle næste position, **c** beholderens centrum, **pc** kollisionspunktet og **p1′** den position, som skal simulere en reflektion udfra væggen.

Kollisionen af en partikel med beholderens væg er illustreret i Fig. 1 Hvis der vil ske en kollision, altså willcollide($\mathbf{p}, \mathbf{v}, \mathbf{c}, r$) returnerer sandt, kan det håndteres i følgende trin:

1. Sæt **p1** = **p**

2. Kollisionspunktet med beholderens væg **pc** kan findes ved at løse ligningen

$$(\mathbf{p1} + \mathbf{v}u - \mathbf{c})^2 = r^2 \tag{9}$$

for skalaren $0 < u \leq 1$, som er en andengrads ligning i u (husk at behandle **p1**, **v** og **c** som vektorer). Find rødderne, vælg den positive rod og sæt

$$\mathbf{pc} = \text{add}(\mathbf{p1}, \mathbf{v}u) \tag{10}$$

3. Vi antager et elastisk sammenstød, således at partiklen bevarer sin kinetiske energi og dermed sin fart, derudover gælder der for elastiske sammenstød at hastighedsvektorens indgangsvinkel er lig udgangsvinklen på den udgående hastighedsvektor efter sammenstødet. Vi kan finde den udadgående vektorretning ved at spejle **p1** over vektoren $\text{vec}(\mathbf{c}, \mathbf{pc})$ (illustreret på Fig. 1 som $\mathbf{p1}'$) og så sætte retningen udfra $\text{vec}(\mathbf{pc}, \mathbf{p1}')$. Dette kan gøres i de følgende trin:
   (a) $\mathbf{vp} = \text{vec}(\mathbf{p1}, \text{proj}(\mathbf{p1}, \text{vec}(\mathbf{c}, \mathbf{pc})))$
   (b) $\mathbf{p1}' = \text{add}(\mathbf{p1}, \text{scale}(\mathbf{vp}, 2\text{len}(\mathbf{vp})))$
   (c) $\mathbf{v2} = \text{scale}(\text{vec}(\mathbf{pc}, \mathbf{p1}'), \text{len}(\mathbf{v}))$
4. Vi kan nu opdatere hastighedsvektoren **v** og positionen **p** af partiklen ved
   (a) $\mathbf{v} = \mathbf{v2}$
   (b) $\mathbf{p} = \text{add}(\mathbf{pc}, \text{scale}(\mathbf{v}, 1 - u))$

   således at partiklen nu befinder sig indenfor beholderens vægge og ved næste tidsskridt vil bevæge sig indad, langs den reflekterede retning.

Nu kan vi gå i gang med opgaven:

(a) Skriv et program, der simulerer det ovenfor beskrevne system (med en enkel beholder). Det er op til jer, hvilket design I vælger, og hvor i programmet tidsskridtene og kollisionstjekket håndteres, men I skal forklare jeres designvalg. I skal desuden sørge for at beholderen initialiseres med nogle rimelige værdier. (Hint: Det skal tage mange tidsskridt for en partikel at bevæge sig gennem beholderen, idet der ellers vil ske kollisioner konstant).

(b) Udvid programmet med funktionalitet til at visualisere systemets tilstand ved et bestemt tidspunkt t. Her skal partiklernes positioner og beholderens væg tegnes (plottes), og information om beholderens radius og antallet af partikler udskrives. Vis resultatet i besvarelsen for et tidspunkt t_i af eget valg. Som alternativ visualisering kan I vælge at lave en animation af simuleringen (se eksempelvis 14ti2).

(c) Gassens *tryk* P i beholderen er i *kinetisk teori* forklaret som stammende fra den kraft, der udøves af gaspartiklerne som kolliderer med beholderens væg. Fra Newton's anden lov har vi at kraften af et legeme er lig masse gange acceleration, d.v.s. på skalar form

$$F = ma\ . \tag{11}$$

Den kraft en gaspartikel påvirker beholderens væg med kan udtrykkes ved partiklens kinetiske energi som er en funktion af partiklens hastighed. Hvis vi antager at der er N partikler i gassen med samme masse m og vores beholder har diameteren $D = 2r$, så kan den samlet kraftpåvirkning opskrives udfra gennemsnitsfarten $|\bar{\mathbf{v}}|$ (dvs. gennemsnitslængden af hastighedsvektorerne) for de N partikler. Trykket i beholderen kan derfor udtrykkes ved

$$P = \frac{F}{D} = \frac{N \cdot m \cdot |\bar{\mathbf{v}}|^2}{2A} \tag{12}$$

for beholderens areal $A = \pi r^2$ (for at fysikken skal passe snyder vi og siger at enhederne på vores "areal"er cm^3 som svarer til at vores todimensionelle cirkel er uendelig tynd og har et volumen som er lig med arealet af cirklen). Implementer ligningen (12) i simulatoren, således at trykket i beholderen kan beregnes udfra partiklernes hastigheder til et hvert tidspunkt i simuleringen.

(d) Idealgasloven siger, simplificeret til vores 2d-simulator, at $P \cdot A = N \cdot k_B \cdot T$, hvor T er systemets temperatur målt i Kelvin $[K]$ og $k_B = 1.380658 \cdot 10^{-23}\ JK^{-1}$ (enhederne er Joule pr. Kelvin) er Boltzmanns konstant. Ved at udnytte (12) har vi så følgende relationer mellem tryk, temperatur og gennemsnitsfart

$$P \cdot A = N \cdot k_B \cdot T = \frac{N \cdot m \cdot |\bar{\mathbf{v}}|^2}{2} \ . \tag{13}$$

Brug ligningen (13) til at udvide simulatoren med funktionalitet til at vi kan angive gassens temperatur T og udfra dette ændre på gaspartiklernes hastighedsvektorer således at den simulerede gas ændre sig til at simulerere gassen ved den nye temperatur T. Dette kræver at vi

(a) beregner en ny gennemsnitshastighed udfra (13).

(b) skalere hastighedsvektorene for de enkelte partikler udfra den nye gennemsnitshastighed. Dette kræver en passende udvidelse af partikelobjektet med funktionalitet til at skalere `v`.

(e) Lav visualiseringer af simulatoren for temperaturen $T = 300\ K$ (i grader celsius svare det til $T = 26.85\ ^\circ C$) og for $T = 373.15\ K$ (svare til $T = 100\ ^\circ C$). Hvis i ikke vælger at lave et animation skal simuleringen køre et vist antal tidsskridt før visualiseringen af partiklernes position produceres.

1.2 Tirsdagsøvelser

Besvarelser af disse opgaver skal ikke afleveres, men opgaverne forventes løst inden torsdag.

14ti1 Vi kan tegne ellipser i `matplotlib` ved at anvende `matplotlib.patches.Ellipse` klassen. Læs dokumentationen for denne klasse for yderligere information og kig evt. på dette eksempel `http://matplotlib.org/examples/pylab_examples/ellipse_demo.html`. Implementer en funktion som kan tegne en cirkel med et givet centrum og radius i et pyplot koordinatsystem.

14ti2 Vi kan lave animationer ved hjælp af matplotlib på mindste to forskellige måder (virker forskelligt på Windows, MacOSX og Linux):

1. `Matplotlib` er som standard i ikke-interaktiv tilstand som betyder at figurere først bliver tegnet ved kald af `pyplot.show()`. Men skifter vi tilstanden til interaktiv ved kald af `pyplot.ion()`, så bliver figurere oprettet med det samme og tegnes ved kald til `pyplot.draw()`. Dette eksempel viser hvad der skal gøres for at lave en simpel animation (denne metode virker på MacOSX og Linux, men ikke på Windows 8):

```
import matplotlib.pyplot as plt
plt.ion()
plt.figure()
x = [-5,-5]
for i in range(100):
    plt.clf() # Slet figuren
    x = [x[0]+i*0.001, x[1]+i*0.001]
    plt.plot(x[0], x[1], '.') # Tegn et punkt
    plt.xlim(-5,5) # Saet begraensninger paa koordinatakserne
    plt.ylim(-5,5) # Saet begraensninger paa koordinatakserne
    plt.draw() # Bed pyplot om at tegne figuren

raw_input("Tryk <retur>")
plt.ioff()
```

2. `Matplotlib` indeholder et animations modul kaldet `animation` som også kan anvendes. Her skal man definere en funktion som tegner grafen for hvert tidsskridt i animationen. Her er et simpelt eksempel som viser animationen på skærmen:

```
import matplotlib
import matplotlib.pyplot as plt
from matplotlib import animation

def drawfigure(i, ax, fig):
   """Funktion som kaldes for hvert tidsskridt"""
   ax.cla() # En anden maade at slette figuren paa
   x = [-5,-5]
   x = [x[0]+i*0.1, x[1]+i*0.1]
   frame = ax.plot(x[0], x[1], '.') # Tegn et punkt
   plt.xlim(-5,5) # Saet begraensninger paa koordinatakserne
   plt.ylim(-5,5) # Saet begraensninger paa koordinatakserne
   return frame

matplotlib.use('Agg')
fig = plt.figure()
ax = plt.subplot(111)
ani = animation.FuncAnimation(fig, drawfigure, frames=xrange(100),
         fargs=(ax, fig), interval=1) # Opret en animation
plt.show() # Vis animation
```

Udvide din løsning til bølgeopgaven fra ugeseddel 12g1, således at du kan animere bølgens bevægelse over tid ved hjælp af en eller begge af disse animationsmetoder.

1.3 Torsdagsøvelser

Besvarelser af disse opgaver skal ikke afleveres, men opgaverne forventes løst inden tirsdag i efterfølgende uge.

14to1 Anvend idealgasloven som angivet i (13) og lav en graf som viser trykket P som funktion af temperaturen T for en beholder med en fast radius r (du vælger selv en værdi).

14to2 Anvend idealgasloven som angivet i (13) og lav en graf som viser trykket P som funktion af beholder radius r for en fast temperatur T (du vælger selv en værdi).

14to3 Anvend idealgasloven som angivet i (13) og lav en graf som viser gennemsnitsfarten $\bar{v}$ (dvs. længden af gennemsnitshastighedsvektoren $\bar{\mathbf{v}}$) som en funktion af temperaturen T.

14to4 Lad os lave en simulator af Jordens bane rundt om Solen. Den type model vi er ved at specificere kaldes en 2-legeme model og vi antager at Solen og Jorden er punktformet masser, dvs. de ikke har nogen udbredelse. Lad os antage at det kun er Jorden der bevæger sig rundt om Solen og at verdenskoordinatsystemet er fastlagt i Solens centrum.

Til enhver tid bliver Jorden påvirket af gravitationskraften fra Solen, hvilket medfører at Jorden accelerere, og denne acceleration kan på vektorform beskrives ved

$$\vec{a} = -\frac{GM_{\text{Solen}}}{r^3}\vec{r}_{Jorden} \ , \tag{14}$$

hvor $\vec{r}_{Jorden}$ angiver vektoren fra Solens centrum til Jordens centrun og $r = \|\vec{r}_{Jorden}\|$. Termen GM_{Solen} er gravitationskonstanten gange Solens masse som har værdien $GM_{\text{Solen}} = 2.959122082322128 \times 10^{-4}$ AU^3/dag^2. Vi kan simulerere Jordens position $\vec{r}_{Jorden}$ ved at opdatere Jordens positions- og hastighedsvektor i tidsskridt Δt

$$\vec{r}(t_{n+1}) = \vec{r}(t_n) + \vec{v}(t_n)\Delta t \tag{15}$$

og

$$\vec{v}(t_{n+1}) = \vec{v}(t_n) + \vec{a}(t_n)\Delta t \ . \tag{16}$$

Vi har altså kun behov for at kende $\vec{a}$, $\vec{v}$ og $\vec{r}$ til tiden t_n, dvs. der vi nåede til i sidste simulationsskridt. Accelerationen $\vec{a}(t_n)$ kan beregnes udfra $\vec{r}(t_n)$ for Jorden ved at anvende (14).

Implementer en simulator af Jordens bevægelse rundt om Solen ved hjælp af ovenstående teori. Du kan benytte følgende vektorer som initielle værdier for positions- og hastighedsvektorene for Jorden

$$\begin{aligned}\vec{r}(t_0) &= (-1.813068419866209 \cdot 10^{-1}, 9.642197733507970 \cdot 10^{-1}, -6.850809238551276 \cdot 10^{-5}) \\ \vec{v}(t_0) &= (-1.718334419397708 \cdot 10^{-2}, -3.209800047122614 \cdot 10^{-3}, 6.736104268755766 \cdot 10^{-9}) \ .\end{aligned}$$

Enheden for positionsvektoren er [AU] (Astronomisk enheder) og for hastighedsvektoren er den [AU/dag] (Astronomisk enheder pr. dag). Du bør anvende et tidsskridt $\Delta t < 1$. Prøv at visualiser Jordens bane som funktion af tiden efter et vist antal tidsskridt.