Prosjekt 3: Modell av solsystemet ved hjelp av ordinære differensialligninger

Live Wang Jensen (`livewj`)
Joseph Knutson (`josephkn`)

October 23, 2016

Abstract

Vi bestemte jordens eksperimentelle unnslipningshastighet til å ligge mellom 8.75 og 8.90 AU/yr, mens den teoretiske verdien var 8.89 AU/yr.

Contents

1 Introduksjon

Målet med dette prosjektet er å utvikle en kode som kan simulere solsystemet ved hjelp av den populære *velocity Verlet* algoritmen. I første del skal vi se på et forenklet solsystem med jorden som eneste planet kretsende rundt solen. Dette gjøres for å teste selve algoritmen og se om alt fungerer. Her skal vi teste både Eulers metode og velocity Verlet metoden, men videre skal vi kun bruke Verlet metoden, da denne er mye mer stabil. Den eneste kraften som virker i systemet vårt er tyngdekraften. Vi innfører deretter Jupiter og observerer hvordan Jupiters masse vil påvirke jordens bane omkring solen. Til slutt skal vi modellere et fullstendig solsystem med alle planeter, og se på hvordan dette oppfører seg som funksjon av tiden. I tillegg skal vi se spesielt på Merkurs bane og hvordan den påvirkes av den relativistiske tyngdekraften. SKRIV MER HER

2 Teori

Det er kun tyngdekraften som virker på systemet vårt. Newtons tyngdelov sier at tyngdekraften som virker mellom to legemer, er proporsjonal med produktet av deres masse, og omvendt proporsjonal med kvadratet av avstanden mellom dem [5]. Dersom vi skal se på tyngdekraften som virker mellom solen og jorden, kan loven kan uttykkes på følgende måte:

$$F_G = \frac{GM_\odot M_{Earth}}{r^2} \tag{1}$$

hvor $M_\odot$ er solens masse, M_{Earth} er jordens masse, G er gravitasjonskonstanten og r er avstanden mellom jorden og solen. Vi antar at massen til solen er mye større enn massen til jorden, slik at vi kan se bort fra tyngdekraften som virker på solen i jord-sol-systemet.

Dersom vi antar at jordens bane rundt solen ligger i xy-planet, kan vi bruke Newtons andre lov til å finne

$$\frac{d^2x}{dt^2} = \frac{F_{G,x}}{M_{\text{Earth}}} \tag{2}$$

og

$$\frac{d^2y}{dt^2} = \frac{F_{G,y}}{M_{\text{Earth}}} \tag{3}$$

hvor $F_{G,x}$ og $F_{G,y}$ er x og y komponentene til tyngdekraften. I dette prosjektet skal vi bruke astronomiske enheter (AU). Èn astronomisk enhet er altså den gjennomsnittlige avstanden mellom jorden og solen, 1 AU = 1.5×10^{11}m. I tillegg bruker vi enheten år i stedet for sekunder til å måle tiden med. I vårt system er solens masse skalert slik at $M_\odot = 1$. Massen til alle de andre planetene skaleres derfor i forhold til solmassen, for eksempel vil $M_{Earth,skalert} = M_\odot/M_{Earth}$.

3 Løsningsmetoder

Vi kan beregne jordens bane rundt solen ved å bruke enten Eulers metode eller velocity Verlet algoritmen til å løse de ordinære differensialligningene gitt ovenfor. Dersom vi antar

at jorden beveger seg i en sirkulær bane rundt solen, vet vi i følge Newtons andre lov at tyngkraften som virker på jorden må være

$$F_G = \frac{M_{Earth}v^2}{r} = \frac{GM_{\odot}M_{Earth}}{r^2}$$

siden $a = v^2/r$ for en sirkulær bane, hvor v er jordens hastighet. Denne ligningen kan brukes til å vise at

$$v^2 r = GM_{\odot} = 4\pi^2 AU^3/yr^2$$

3.1 Euler

Vi kan diskretisere ligningene gitt ovenfor slik at vi kan sette opp en algoritme for å løse ligningene ved hjelp av Eulers metode. Dersom vi har en funksjon y, kan vi finne ut hvordan funksjonen ser ut i et tidssteg y_{i+1} ved hjelp av det tidligere tidssteget y_i:

$$y_{i+1} = y(t = t_i + h) = y(t_i) + h\Delta(t_i, y_i(t_i)) + O(h^{p+1})$$

hvor $h = (b-a)/N$ er steglengden, N er antall tidssteg, a og b er integrasjonsgrensen og $O(h^{p+1})$ er feilen i beregningen vår. Δ representerer alle de deriverte av funksjonen vår y:

$$\Delta(t_i, y_i(t_i)) = (y'(t_i) + ... + y^{(p)}(t_i)\frac{h^{p-1}}{p!})$$

Hvis vi definerer

$$y'(t_i) = f(t_i, y_i)$$

og minker Δ til å kun inneholde den førstederiverte av y, får vi

$$y_{i+1} = y(t_i) + hf(t_i, y_i) + O(h^2)$$

og

$$t_{i+1} = t_i + h$$

som sammen utgjør Eulers metode. Legg merke til at vi for hvert tidssteg får en feil på $O(h^2)$. Dette gir oss en global feil på $N \cdot O(h^2) \approx O(h)$ [7]. Eulers metode er altså ikke å anbefale hvis vi ønsker nøyaktighet i beregningene, men kan være nyttig dersom vi ønsker et førsteintrykk av hvordan løsningen kan se ut.

Et eksempel på hvordan dette kan implementeres i C++ vises i figur 1. Her er `Euler` en funksjon i klassen med samme navn. Funksjonen starter med å hente funksjonen `calculateForcesAndEnergy()` fra klassen vår `SolarSystem`, som inneholder alle de nødvendige metodene for å simulere solsystemet, bortsett fra integrasjonsmetodene. Inne i for-løkken beregnes legemenes posisjon og hastighet ved hjelp av Eulers algoritme.

```
void Euler::integrateOneStep(SolarSystem &system)
{
    system.calculateForcesAndEnergy();

    for(CelestialBody &body : system.bodies()) {
        body.position += body.velocity*m_dt;
        body.velocity += body.force / body.mass * m_dt;
    }
}
```

Figure 1: Eksempel på hvordan Eulers algoritme kan implementeres i en klasse i C++. Koden er hentet fra filen `euler.cpp`.

3.2 Velocity Verlet

En annen metode å løse differensialligningene på er Verlet metoden. Denne populære algoritmen er både numerisk stabil og enkel å implementere. Dersom vi tar for oss Newtons andre lov som en andre ordens differensialligning, som i én dimensjon kommer på formen

$$m\frac{d^2x}{dt^2} = F(x,t)$$

som kan skrives om til to *koblede* differensialligninger

$$\frac{dx}{dt} = v(x,t) \quad og \quad \frac{dv}{dt} = F(x,t)/m = a(x,t)$$

Dersom vi nå gjør en Tayorutvikling

$$x(t+h) = x(t) + hx^{(1)}(t) + \frac{h^2}{2}x^{(2)}(t) + O(h^3) \tag{4}$$

I vårt tilfelle kjenner vi den andrederiverte via Newtons andre lov, $x^{(2)} = a(x,t)$. Dersom vi legger til Taylorutviklingen for $x(t-h)$ i ligning 4, ender vi opp med

$$x_{i+1} = 2x_i - x_{i-1} + h^2x_i^{(2)} + O(h^4)$$

hvor $x_{i\pm1} = x(t_i \pm h)$ og $x_i = x(t_i)$. Feilen her går som $O(h^4)$, siden alle de odde leddene kansellerer i addisjonen [7]. Hastigheten er ikke direkte inkludert i ligningen, siden $x^{(2)}$ er kjent. Dersom vi trenger hastigheten, kan vi finne den ved å bruke ligningen

$$x_i^{(1)} = \frac{x_{i+1} - x_{i+1}}{2h} + O(h^2)$$

Hastigheten har altså en feil på $O(h^2)$. Legg merke til at algoritmen for posisjonen x for $i = 0$ trenger den ikke-eksisterende verdien x_{-1} for å starte. Dette problemet unngås i **velocity Verlet** metoden. Taylorutviklingen til posisjonen er gitt som

$$x_{i+1} = x_i + hx_i^{(1)} + \frac{h^2}{2}x_i^{(2)} + O(h^3)$$

mens Taylorutviklingen til hastigheten er

$$v_{i+1} = v_i + hv_i^{(1)} + \frac{h^2}{2}v_i^2 + O(h^3)$$

Ved hjelp av Newtons andre lov har vi et analytisk uttrykk for den deriverte til hastigheten, nemlig

$$v_i^{(1)} = \frac{d^2x}{dt^2}|_i = \frac{F(x_i, t_i)}{m}$$

Dersom vi adderer dette til Taylorutviklingen til akselerasjonen

$$v_{i+1}^{(1)} = v_i^{(1)} + hv_i^{(2)} + O(h^2)$$

og ser bort fra de høyere ordens leddene etter den andrederiverte av hastigheten, får vi

$$hv_i^{(2)} \approx v_{i+1}^{(1)} - v_i^{(1)}$$

Vi kan nå skrive om Taylorutviklingen til hastigheten som

$$v_{i+1} = v_i + \frac{h}{2}\left(v_{i+1}^{(1)} + v_i^{(1)}\right) + O(h^3)$$

Det vi ender opp med til slutt er den berømte **velocity Verlet** algoritmen:

$$x_{i+1} = x_i + hv_i + \frac{h^2}{2}v_i^{(1)} + O(h^3)$$

$$v_{i+1} = v_i + \frac{h}{2}\left(v_{i+1}^{(1)} + v_i^{(1)}\right) + O(h^3)$$

Implementasjonen finnes i filen `velocityverlet.cpp`, og et lite utsnitt av denne algortimen vises i figur 2. Funksjonen `VelocityVerlet` starter med å initialisere kreftene som virker på systemet ved hjelp av funksjonen `calculateForcesAndEnergy`, før den bruker velocity Verlet metoden til å beregne legmenes hastighet og posisjon. Etter at dette er gjort, oppdateres kreftene før hastigheten beregnes igjen.

```
void VelocityVerlet::integrateOneStep(SolarSystem &system)
{
    //initialize forces
    if (first) {
        system.calculateForcesAndEnergy();
        first = false;
    }

    for(CelestialBody &body : system.bodies()) {
        body.velocity += body.force / body.mass * m_dt * 0.5;
        body.position += body.velocity*m_dt;
    }

    system.calculateForcesAndEnergy(); //update forces

    for(CelestialBody &body : system.bodies()) {
        body.velocity += 0.5*(body.force/body.mass)* m_dt;
    }
}
```

Figure 2: Eksempel på hvordan velocity Verlet algoritmen kan implementeres i en klasse i C++. Koden er hentet fra filen `velocityverlet.cpp`.

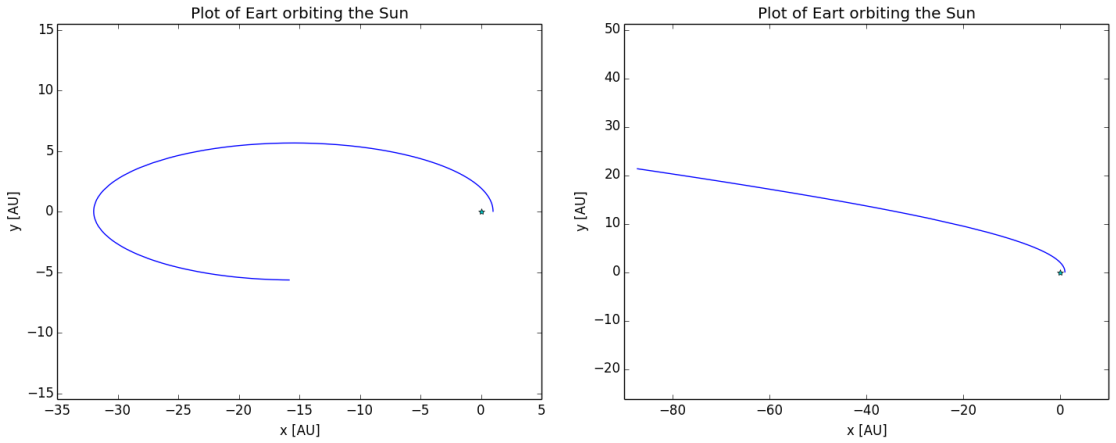
4 Resultater

4.1 Test av algoritmen

oppg 3c

4.2 Unnslipningshastighet

Ved hjelp av prøving og feiling, fant vi unnslipningshastigheten til jorden i en avstand 1 AU fra solen. Dette ble gjort ved å sette jordens posisjon lik $(1, 0, 0)$ og hastighet $(0, v_y, 0)$. Ved å variere verdien til v_y fant vi at unnslipningshastigheten må ligge et sted mellom v_y = 8.75 AU/yr og v_y = 8.90 AU/yr, se figur 3.



(a) v_y = 8.75 AU/yr (b) v_y = 8.90 AU/yr

Figure 3: Figurene viser jordens bane for ulike unnslipningshastigheter. Vi ser at denne hastigheten må ligge et sted mellom 8.75 og 8.90 AU/yr. I dette tilfellet er det ingen krefter som virker på solen, og den er markert med en stjerne i origo.

Vi kan finne et eksakt svar ved å bruke formelen for unnslipningshastigheten:

$$v_e = \sqrt{\frac{2GM}{r}}$$

Her er G tyngdeakselerasjonen, M solens masse og r avstanden mellom solen og jorden, som i dette tilfellet er 1 AU. Massen til solen er $M_\odot = 1$, og vi ender opp med

$$\sqrt{\frac{2 \cdot 4\pi^2 \cdot 1}{1}} \approx 8.89 \quad AU/yr$$

4.3 Trelegemeproblemet

(oppg 3e)

4.4 Endelig modell med alle planeter i solsystemet

3f

4.5 Merkurs perihelpresesjon

Presesjon er et fysisk fenomen som vil vise seg når aksen til et roterende objekt "slingrer" mens det utsettes for en ytre kraft. Dette viser seg å gjelde for planetene i solsystemet

vårt også. Vi kan se på presesjonen som en langsom endring i rotasjonsaksens retning for et legeme som roterer og samtidig påvirkes av en ytre kraft, i vårt tilfelle tyngdekraften. I virkeligheten er ikke en planets bane rundt solen ellipseformet. Dersom vi lar planeten gå mange nok runder rundt sola, vil vi kunne se at banen heller er kronblad-formet. Dette kommer av av aksen til planetens elliptiske bane preseserer, slik at vi får en langsom endring i rotasjonsaksen. Avvik mellom den observerte størrelsen på perihelpresesjonen for Merkur og den spådd av den klassiske mekanikken var fremtredende blant de former for eksperimentelle bevis som førte til aksept av Einsteins relativitetsteori (spesielt hans generelle relativitetsteori) som nøyaktig spådde anomaliene

5 Diskusjon

Den eksperimentelt funnede unnslipningshastigheten stemmer godt overens med den eksakte løsningen. Da vi anslo at v_e måtte ligge i intervallet <8.75, 8.90>, fikk vi en eksakt løsning på 8.89 AU/yr.

6 Vedlegg

Alle koder og resultater som er brukt i rapporten finnes på Github-adressen: https://github.com/livewj/PROJ

7 Konklusjon

Samspillet mellom den eksperimentelle og den eksakte unnslipningshastigheten viser at vår modell av solsystemet ser ut til å fungere.

References

[1] Kursets offisielle Github-side *FYS3150 - Computational Physics* https://github.com/CompPhysics/ComputationalPhysics, 19.10.2016

[2] M. Hjort-Jensen: Computational physics, lecture notes 2015. Fysisk institutt, UiO, 2016.

[3] Oppgavetekst: Project 3, Fysisk institutt, UiO, 19.10.16

[4] "solar-system-fys3150" https://github.com/mortele/solar-system-fys3150 19.10.2016

[5] Tyngdekraft https://no.wikipedia.org/wiki/Tyngdekraft 19.10.2016

[6] HORIZONS Web-Interface: http://ssd.jpl.nasa.gov/horizons.cgi#top 13.10.2016

[7] Slides fra kursets offisielle nettside "Ordinary differential equations": http://compphysics.github.io/ComputationalPhysics/doc/pub/ode/pdf/ode-beamer.pdf, 21.10.16

[8] "Presesjon" https://no.wikipedia.org/wiki/Presesjon, 21.10.16