Tieteellinen laskenta 2 lopputyö: Planetary motion simulator

Vili Oja 014339369

20. joulukuuta 2015

1 Esittely

Mikäli haluamme tutkia n-kappaleen systeemin evoluutiota, simulointi on ainoa tapa, sillä tällaiseen tilanteeseen ei ole olemassa analyyttistä ratkaisua. Siksi planeettojen liikkeiden simulointi on erittäin hyödyllistä esimerkiksi tutkittaessa aurinkokuntien kehitystä.

Taivaankappaleiden tulevien sijaintien tietäminen on hyödyllistä esimerkiksi taivaan havainnoinnin ja avaruudentutkimuksen kannalta. Kun luotamia lähetetään pitkille matkoille, haluamme useimmiten käyttää jotain planeettaa gravitaatiolinkona, ja jotta tämä onnistuisi meidän täytyy tietää tarkkaan planeettojen paikat tulevaisuudessa, jotta pystymme lähettämään luotaimen oikeaan paikkaan oikeaan aikaan. Lisäksi, mikäli aurinkokuntaamme ilmestyy jokin entudestaan tuntematon asteroidi tai komeetta, niiden radan määrittäminen onnistuu parhaiten simulaatiolla.

Jos havaitsemme kaukaisia erittäin nuoria aurinkokuntia jotka eivät ole vielä lopullisessa tilassaan, vaan ovat vasta kehittymässä, voimme simuloida myöskin niiden kehitystä.

2 Menetelmät

Käytin simulaattorissani neljänennen kertaluvun Runge-Kutta integrointia (RK4).

Tarkastellaan muuttujien $r = r(t)$ ja $v = v(t)$ kytkettyä tavallisten differentiaaliyhtälöiden ongelmaa

$$\begin{aligned} \dot{r} &= f(r, v) \ , \\ \dot{v} &= g(r, v) \ , \end{aligned} \tag{1}$$

missä funktiot $f = f(r, v)$ ja $g = g(r, v)$ ovat mielivaltaisia.

Neljännen kertaluvun Runge-Kutta menetelmä kuvaa alkuarvot (r_0, v_0) yhden aika-askeleen τ eteenpäin seuraavasti:

$$\begin{aligned} r_1 &= r_0 + \frac{1}{6}\tau(a_r + 2b_r + 2c_r + d_r) \\ v_1 &= v_0 + \frac{1}{6}\tau(a_v + 2b_v + 2c_v + d_v) \end{aligned} \tag{2}$$

missä

$$\begin{aligned} a_r &= f(r_0, v_0) \\ b_r &= f(r_0 + \frac{1}{2}\tau a_r, v_0 + \frac{1}{2}\tau a_v) \\ c_r &= f(r_0 + \frac{1}{2}\tau b_r, v_0 + \frac{1}{2}\tau b_v) \\ d_r &= f(r_0 + \tau c_r, v_0 + \tau c_v) \end{aligned} \tag{3}$$

ja

$$\begin{aligned} a_v &= g(r_0, v_0) \\ b_v &= g(r_0 + \frac{1}{2}\tau a_r, v_0 + \frac{1}{2}\tau a_v) \\ c_v &= g(r_0 + \frac{1}{2}\tau b_r, v_0 + \frac{1}{2}\tau b_v) \\ d_v &= g(r_0 + \tau c_r, v_0 + \tau c_v)\ . \end{aligned} \tag{4}$$

Koska haluamme siis simuloida painovoiman vaikutusta n kappaleelle, meidän täytyy valita sopivat funktiot f ja g. Ensimmäinen funktio kertoo paikan muutoksen, mikä on tietenkin vain nopeus, joten

$$\dot{\bar{r}}_i = \bar{v}_i\ . \tag{5}$$

Toinen funktio, eli nopeuden muutos, puolestaan on painovoiman aiheuttama kiihtyvyys, ja tämä lasketaan seuraavasti:

$$\begin{aligned} \dot{\bar{v}}_i &= -\gamma \sum_{j=1, j\neq i}^{n} m_j \frac{\bar{r}_i - \bar{r}_j}{r_{ij}^3}\ , \\ r_{ij} &= |\bar{r}_i - \bar{r}_j|,\ j = 1, ..., n\ . \end{aligned} \tag{6}$$

Nyt siis $\bar{r}_i$ ja $\bar{v}_i$ ovat vektoreita, jotka sisältävät yhden kappaleen paikat ja nopeudet.

Tämän menetelmän avulla voimme simuloida n:nää kappaletta mielivaltaisen ajanjakson eteenpäin.

3 Implementaatio

3.1 Ohjelmien toteutus

Kirjoitin simulaattorin tiedostoon rk4.f90 ja siihen liittyviä funktioita moduuliin funktiot.f90.

Pääohjelma ensin lukee sille komentoriviargumentteina annetut tiedot, eli tiedosto alkudatalle, tiedosto, johon simuloitu data tallennetaan, simuloitavan ajanjakson pituus, datan tallennustiheys sekä aika-askeleen pituus. Ohjelma lukee määritellystä tiedostosta kuinka monta kappaletta on simuloitavana, ja luo tämän tiedon pohjalta sopivan kokoiset arrayt kappaleiden tietojen säilytykseen.

Sitten ohjelma suorittaa itse laskemisen ja datan kirjaamisen. Kaikki tapahtuu yhden loopin sisällä, joka pyörii kunnes simulaatio on edennyt halutun pituisen ajanjakson. Loopissa laketaan Runge-Kutta menetelmän antamien kertoimien avulla kaikkien kappaleiden uudet paikat ja nopeudet, jotka yhden kieroksen lopuksi talletettaan kappaleiden uusiksi arvoiksi. Mikäli kierros on myös sellainen, jolloin paikat kirjataan muistiin, ohjelma muodostaa kappaleiden paikoista rivin dataa, ja kirjoittaa sen spesifioituun tiedostoon.

Moduulissa olen määritellyt itse RK4:ään liittyviä funktioita. Siellä on funktiot 'f' ja 'g', eli siis paikan ja nopeuden muutokset. Paikan muutos palauttaa vain sille syötetyn nopeuden. Nopeuden muutos laskee kaavassa 6 nähtävän funktion avulla nopeuden muutoksen. Käytännössä tämä tehdään loopilla, jossa aina yhdellä kierroksella lasketaan yhden toisen kappaleen gravitaation vaikutus käsiteltävään kappaleeseen. Tätä summaa varten tein myös funktion vektorin pituuden laskemiseen, joka sijaitsee myöskin tässä moduulissa.

Viimeiseksi moduulissa on funktio itse RK4-menetelmän määräämän kahdeksan kertoimen laskemiseksi. Näitä laskettaessa käytetään hyväksi paikan ja nopeuden muutoksen laskemiseen tekemiäni funktioita. Lopuksi vektorimuotoiset kertoimet säilötään muuttujaan, jonka funktio palauttaa.

Itse numeerisen integraattorin lisäksi tein pythonilla ohjelman, joka tekee tallentamastani datasta kuvia. Tein tästä piirtäjästä useita eri versioita tehtävänannon määräämien tilanteiden mukaisiin tarkoituksiin.

Kaikissa piirtäjissa perusrakenne on seuraava: Ohjelma lukee komentoriviltä tiedoston nimen, jossa haluttu data on tallennettu, sekä kuinka monen päivän välein laskettaessa

data tallennettiin. Tätä käytetään kuvia piirrettäessä ajankohdan kirjaamiseen. Ohjelma laskee annetun tiedoston datasta kuinka monta kappaletta siinä on simuloituna, joten käyttäjän ei tarvitse välittää tätä tietoa ohjelmalle itse.

Ohjelma piirtää kuvan, jossa on piirtäjästä riippuen kaksi tai neljä kuvaa. Koko aurinkokunnan simulaation piirtämiseen tarkoitetu versio tekee neljä kuvaa: ensimmäisessä kahdessa niistä näkyy lähikuva planeetoista Merkuriuksesta Marsiin, sekä xy- että yz-tasossa, ja lopussa kahdessa näkyy koko aurinkokunta samoin sekä xy- että yz-tasossa. Muissa piirtäjän versioissa näkyy vain kaksi kuvaa: kyseiseen tapaukseen sopivan etäisyyden kuva sekä xy- että yz-tasossa.

Piirtämisen jälkeen ohjelma tallentaa kuvan nimellä 'kuva[numero].png', missä numero etenee välillä 0000-9999. Näistä kuvista on helppo muodostaa vaikkapa gif-muotoisia animaatioita.

Piirtäminen tapahtuu matplotlib-kirjaston pyplot-käyttöliittymän avulla, joka on datan piirtämiseen tarkoitettu kirjasto, tarjoten samanlaisia toimintoja kuin MATLABin piirtämistyökälut.

3.2 Ohjelmien käyttö

Käyttökelpoisen ohjelman luominen onnistuu helposti tekemäni makefile:n avulla. Eli täytyy vain navigoida terminaalilla kansioon src/ ja siellä ajaa seuraavat komennot:

make
make clean

'make clean' ei ole pakollinen, mutta se poistaa kääntämisen yhteydessä syntyvät käyttäjälle turhat tiedostot. Nyt kansioon on luotu ohjelma rk4.exe. rk4.exe sekä kaikki eri piirtäjät kannattaa siirtää kansioon run/ , jotta ne löytävät niiden tarvitsemat tiedostot.

Ohjelmien suorittaminen tapahtuu kansiossa run/ (tai missä tahansa muussa kansiossa, johon on siirretty sekä alkudatan sisältävät tiedostot ja käytettävät ohjelmat). Ohjelman rk4.exe suoritus tapahtuu seuraavilla komentoriviargumenteilla:

./rk4.exe [inpufile] [outputfile] [aikajakso] [tallennustiheys] [aika-askel]

Inputfile ja outputfile kertovat mistä tiedostosta ohjelma lukee simuloitavien kappaleiden alkuperäiset ominaisuudet, ja mihin simuloitu data tallennetaan. Inputfilejä on valmiina

neljä: aurinkokunta.txt, aurinko-jupiter.txt, aurinko-maa.txt ja maa-kuu.txt. Tiedostojen nimet kertovat minkä kaikkien kappaleiden tiedot tiedostoissa on. Aikajakso on vuosissa se aika, joka halutaan simuloida. Tämä voi olla kokonaisluku tai desimaaliluku. Tallennustiheys kertoo ohjelmalle kuinka monen aika-askeleen välein uudet lasketut paikat tallennetaan. Täytyy olla kokonaisluku. Mikäli tämä on 1, tallennetaan jokainen simuloitu paikka. Aika-askel kertoo kuinka monen päivän välein ohjelma laskee uudet paikat. Voi olla kokonaisluku tai desimaaliluku. Eli jos tämä on 1, ohjelma laskee uudet paikat kerran päivässä. Jos tämä on 0.5, ohjelma laskee uudet paikat kaksi kertaa päivässä.

Esimerkki tästä:

./rk4.exe aurinkokunta.txt aurinkokunta_10v.dat 10 30 1

Piirtäjiä kutsutaan seuraavalla komennolla:

python piirtaja_[tyyppi].py [inpufile] [aikaväli]

Tässä tyyppi tarkoittaa sitä, millaista piirtäjää halutaan käyttää. Piirtäjiä on neljä: piirtaja_aurinkokunta.py, piirtaja_aurinko-jupiter.py, piirtaja_aurinko-maa.py ja piirtaja_maa-kuu.py. Nimet kertovat millaisten systeemien kuvantamiseen ne on tarkoitettu. Inputfile kertoo mistä tiedostosta piirrettävä data katsotaan. Käytännössä sama kuin rk4:n outpufile. Aikaväli kertoo kuinka monen päivän välein inputfilen paikat on tallennettu. Tätä käytetään ajanhetken kirjaamiseen kuviin. Voi olla kokonaisluku tai desimaaliluku. Käytännössä tämä on sama kuin rk4:n [tallennustiheys] * [aika-askel].

Esimerkki tästä:

python piirtaja_aurinkokunta.py aurinkokunta_10v.dat 30

4 Tulokset

Kaikkien tässä ksäiteltävien tapausten animaatiot on nähtävillä kansiossa 'valmiit kuvat'. Ne on nimetty siten, että pitäisi olla selkeää mikä kuva liittyy aina mihinkin kohtaan.

4.1 Jupiterin simulointi

Laitoin simulaation pyörimään 12 vuoden ajan, joka on hieman yli Jupiterin kiertoaika. Kokeilin aluksi laskea uusi arvoja päivän välein, mikä tuotti hyviä tuloksia. Jupiterin kiertoaika on noin 11,86 vuotta, ja tekemieni kuvien mukaan hetkellä T = 11,863 Jupiter on hyvin tarkasti lähtöpaikassaan. Nostin aika-askelta 30 päivään ja lopulta 100 päivään. 30 päivän laskemisella Jupiter on hyvin lähellä lähtöpaikkaansa hetkellä T = 11,836. 100 päivän välein mitatessa Jupiter ehtii hypätä alkupaikkansa yli, mutta lähimmät arvot paikan molemmin puolin ovat T = 11,78 ja T = 12,05 , joten tällöinkin kiertoaika on vielä hyvin lähellä todellista.

Katsomalla simulaation aika-askel=1 tuloksia, Jupiter näyttää menevän hyvin tarkalleen 11,86 vuodessa täyden kierroksen.

Tekemissäni piirtäjissä origo keskittyy aina keskuskappaleeseen, joten Auringon liikettä ei voi havaita sillä tehdyistä kuvista. Muokkaamalla piirtäjää ja tekemällä sillä kuvan, sain aikaan kuvan aurinko-jupiter_12v_eikeskitetty.gif, jossa origo pysyy Auringon alkupisteessä. Tässäkään ei kuitenkaan ole havaittavissa juuri mitään liikettä Auringon osalta. Jos katsomme laskettuja arvoja, näemme että Aurinko on liikkunut tässä ajassa noin 0,03 AU:ta x-suunnassa ja 0.00003 AU:ta y-suunnassa.

4.2 Yleisiä ajanjaksoja

Simuloin vuoden ajan Maan liikettä Auringon ympäri ja kuukauden ajan Kuun liikettä Maan ympäri. Käytin Aurinko-Maa tapauksessa aika-askeleena yhtä päivää, ja Maa-Kuu tapauksessa puolta päivää. Molemmissa tapauksissa systeemit pyyivät hyvin kasassa.

4.3 Koko aurinkokunta

Simuloin koko aurinkokuntaa Auringosta Uranukseen aluksi 2 vuoden yli. Jos aika-askel oli 100 tai 30 päivää, sisäplaneetat linkoutuivat melko nopeasti ulos. Kun laskin aika-askeleen yhteen kertaan päivässä, aurinkokunta pyöri kunnolla. Simuloin tällä aika-askeleella aurinkokuntaa 200 vuoden ajan, ja tein tästä simulaatiosta videon.

5 Johtopäätökset

RK4-menetelmällä tehty numeerinen integraattori on erittäin tehokas ja nopea laskemaan pitkiäkin ajanjaksoja. Sillä sai aikaan tarkkoja arvioita planeettojen paikoille ja piirtäjällä kivoja animaatioita.

Pythonilla piirtämisen yksi heikkous on kuitenkin se, että se ei ole kovin nopeaa. Esimerkiksi 200 vuoden animaation piirtämisessä kesti hyvä tovi kun ohjelma joutui luomaan yli 3000 kuvaa.

Laskiessani planeettojen alkuperäisiä tietoja en ottanut huomioon niiden inklinaatioita. En kuitenkaan usko, että tämä muutti simulaatiota dramaattisesti, sillä aurinkokunnassamme kaikki kappaleet ovat erittäin lähellä ekliptikaa, eikä mikään niistä liiku erittäin kaltevalla radalla. Koodia tai kuvitusohjelmaa ei kuitenkaan tarvitsisi muuttaa kolmannen ulottuvuuden huomioon ottamiseksi, sillä se on jo mukana koodissa. Ainoa asia mitä tarvitsisi tehdä, on antaa kappaleille realistiset aloituspaikat ja -nopeudet.