Orbits.py

”Yksinkertainen” kiertoratalaskuri

Markus Koskimies
mkoskim@gmail.com
http://mkoskim.drivehq.com

Versio 0.1

Sisällysluettelo

1 Käyttöohjeet

Kiertoratalaskurin käyttöohjeita.

1.1 Asentaminen

Hae verkkosivulta orbits.tar.gz -paketti ja pura se omaan hakemistoon.

1.2 Käynnistäminen

Käynnistä ensin komentoriviltä python:

```
$ python
>>>
```

Tämän jälkeen ladataan python-tulkkiin sisälle kiertoratalaskurin funktiot:

```
>>> from orbits import *
```

Samaan syssyyn voi ladata myös aurinkokunnan (vailinaiset) datat:

```
>>> from solsystem import *
```

Tämän jälkeen ei tarvitse muistella ulkoa Aurinkokunnan kappaleiden parametreja (koko, etäisyydet jne), mutta kuten sanottu, tietokanta ei vielä sisällä kaikkea.

1.3 Omien komentojonojen tekeminen

Seuraavissa harjoituksissa komennot on syötetty käsin python-tulkkiin, johon on ensin ladattu import-direktiivillä radanlaskentapaketti.

Samat komennot voi myös kirjoittaa tiedostoon ja ajaa tulkista komennolla:

```
>>> from MunKomennot import *
...
```

Myös omien ohjelmistojen tekeminen on helppoa. Lisää alkuun ”from orbits import *”, jonka jälkeen voit kirjoitella oman ohjelman. Sen voi käynnistää komennolla (komentoriviltä):

```
$ python MunSofta.py
```

Hauskaa koodailemista!

1.4 Harjoituksia

Ensimmäisenä kannattaa hakea käsiinsä Juhani Kaukorannan Raahen lukiolle tekemä ”avaruusfysiikkaa.pdf” - käytetään tätä harjoitusopuksena:

http://koti.mbnet.fi/jukaukor/fysiikka/avaruusfysiikkaa.pdf

Aloitetaan. Ensimmäisenä Kaukoranta selvittää rakettimoottorin perusteita, mutta nykyinen ohjelmisto ei vielä tue niitä. Sen ratalaskennassa on vielä puutteita, erityisesti erilaisten missioiden tekemisessä (tarkoitus olisi, että ohjelmisto tukisi monivaiheisia avaruusmissioita; useampivaiheisilla, erilaisilla moottoreilla varustetuilla raketeilla).

Harjoitus 1. Mene sivulle 8: Keplerin lait. Ohjelmistoon on sisään rakennettuna Keplerin 3. lain ratkaiseminen. Maan kiertoaika Auringon ympäri on 365.25 päivää ja etäisyys 1 AU. Jos Jupiterin etäisyys on 5.2 AU, niin paljonko on sen kiertoaika?

```
>>> solve_aPaP(1, 365.25, 5.2, None)
4331.0703697815852
```

Koska kiertoaika on annettu päivissä, myös vastaus tulee päivissä. Helpoimmalla päästään, kun käytetään pythonia laskimena:

```
>>> solve_aPaP(1, 365.25, 5.2, None)/365
```

11.865946218579685

Tarkastetaan sopivasta lähteestä ja havaitaan, että hyvin toimii.

Harjoitus 2. Ratalaskennassa tarvitaan hyvin usein keskuskappaleen massaa, mutta ei kilogrammoina, vaan valmiiksi gravitaatiovakiolla kerrottuna (nk. GM). Ohjelmistossa on valmiina tämän ratkaiseminen, jos tunnetaan satelliitin kiertoaika (periodi) ja etäisyys. Jos kiertoaika annetaan sekunteina ja etäisyys metreinä, saadaan melko standardin mukainen GM.

Lasketaan Auringon GM. Oletetaan, että Maapallo on paljon sitä kevyempi. Maapallon kiertoaika on 365.25 päivää ja etäisyys 1 AU (149.6 miljoonaa kilometriä). Ohjelmistossa on valmiina funktioita konversioita varten (voit tarkastaa laskennan oikeellisuuden Kaukorannan kalvoista, sivu 10):

```
>>> solve_GM_from_aP(149.6e9,365.25*24*60*60)
1.3272312023850498e+20
>>> solve_GM_from_aP( AU2m(1), 365.25*24*60*60)
1.3272312023850498e+20
```

Hyvältä näyttää. Lasketaan Kaukorannan kalvoilla sivulla 12 oleva esimerkki, Marsin massan laskeminen. Ensin lasketaan GM ja se muutetaan kilogrammoiksi valmiina olevalla vakiolla (const_G):

```
>>> solve_GM_from_aP(9380e3,time_hms(7,39.5,0))/const_G
6.4239126732932665e+23
```

Edelleen menee oikein.

Harjoitus 3. Seuraavaksi Kaukoranta selittää energian säilymisperiaatetta. Sen varaan ohjelmisto on vähintään puolittain rakennettukin. Sivulla 19 alkaa sitten varsinainen ratalaskenta. Ohjelmistoon on jo valmiiksi täytetty tietoja Maasta, Kuusta, Marsista ja Venuksesta.

Katso sivun 19 laskelmia. Toistetaan ne. Tällä ratalaskimella yleensä ottaen ei puuhata suoraan kaavojen kanssa, vaan tehdään ratoja, orbitaaleja. Näiden yhteyteen tallennetaan radasta erilaisia tietoja.

Tehdään ensin lyhennysmerkintä:

```
>>> Earth = masses[”Earth”]
```

Ympyrärata Maapallon kiertoradalla, 300 km korkeudella:

```
>>> o = Orbit_altitude(Earth, 300e3)
```

Tehtiin rata (muuttujaan o) Maapallon pinnasta 300 kilometrin korkeuteen. Voidaan tarkastella sen ominaisuuksia:

```
>>> print o.a()              # Puoliakselin pituus, a
6678000.0                    # Maapallon säde (r) 6378000 m + 300000 m
>>> print o.v(o.a())         # Ratanopeus korkeudella a
7725.83947914                # m/s
```

1000 kilometrin korkeudella:

```
>>> o = Orbit_altitude(Earth, 1000e3)
>>> print o.v(o.a())
7350.20687062
```

GEO (geosynkroninen rata), 35790 kilometrin korkeudella:

```
>>> o = Orbit_altitude(Earth, 35790e3)
>>> print o.v(o.a())
3074.52045131
>>> print o.P()              # Periodi eli kiertoaika
86175.8320458                # sekunteina
>>> print o.P()/3600
23.9377311238                # tunteina
```

Harjoitus 4. (kalvot, s. 19) Satelliitti kiertää ympyrärataa 300 kilometrin korkeudella. Sille annetaan 200 m/s lisänopeus. Millaiselle elliptiselle radalle satelliitti siirtyy?

```
>>> o = Orbit_from_circular(Earth,Earth.radius + 300e3,200)
>>> print o.r1, o.r2              # Tulostetaan perigeum ja apogeum
6678000.0 7417215.8976
```

Alin kohta (perigeum) pysyy paikallaan, korkein kohta (apogeum) siirtyy 7417 kilometrin korkeuteen (kuten pitikin).

Harjoitus 5. Hämäläinen heittää kiveä Deimoksella – Deimoskin löytyy nykyisestä tietokannasta. Lähtönopeudeksi annetaan v0 = 4.5 m/s. Laskettava radan alin kohta:

Tehdään lyhennysmerkintä:

```
>>> Deimos = masses[”Deimos”]
```

Sitten tehdään rata:

```
>>> o = Orbit_from_v(Deimos, Deimos.radius,4.5)
>>> print o.r1,o.r2
6000.0 6140.82969432
```

Radan periapsis = 6000.00 km, apoapsis = 6140.83 km (Deimoksen säteeksi tallennettu 6 kilometriä). Jaa, Kaukorannalla on jotkut muut mitat. Tehdään uusi Deimoksen näköinen massa Kaukorannan parametreilla ja samanlainen rata kuin äskettäinkin:

```
>>> Deimos2 = Mass(None,"Deimos2", kg2GM(1.8e15),7490,None)
>>> o = Orbit_from_v(Deimos2, Deimos2.radius,4.5)
>>> print o.r1,o.r2
7490 12830.5049858
>>> o.r2 – o.r1
5340.5049849999996
```

Hyvä, samat tulokset kuin Kaukorannalla. Radan korkeimman kohdan nopeuden laskenta (kts. kalvot s. 21):

```
>>> o.v(o.r2)
```

```
2.6269425901201591
```

Funktio v() siis palauttaa nopeuden suhteessa säteeseen, kuten ellipsiradoilla kuuluukin. Radan periodi:

```
>>> o.P()/3600
5.1576316301540519
```

Eli edelleen pysytään hyvin kartalla.

Harjoitus 6. Nyt hämäläinen kapuaa 50 metrin korkuiselle vuorelle ja heittää sieltä pesäpalloaan. Toisella puolella se hipoo Deimoksen pintaa. Tällainen rata tehdään helposti ellipsiradasta:

```
>>> o = Orbit_elliptical(Deimos2,Deimos2.radius+50,Deimos2.radius)
```

Kuinka suuri pitää olla pallon lähtönopeus? Ohjelmisto tallentaa aina ensimmäisen säteen muuttujaan r1 ja toisen muuttujaan r2, joten tässä tapauksessa kysytään, paljonko on nopeus säteen r1 etäisyydellä (apoapsis):

```
>>> o.v(o.r1)
3.9845065177704875
```

Mikä on nopeus Deimoksen toisella puolella (periapsis)?

```
>>> o.v(o.r2)
4.0111053596781678
```

Mikä on radan periodi?

```
>>> o.P()/3600
3.2808550758277808
```

Harjoitus 7. Avaruusaluksen laskeutuminen (kalvot s. 23). Alus halutaan ympyräradalta (alt = 300 km) ellipsiradalle, jonka perigeum on Maapallon pinnalla. Vaikka ratalaskimessa on jo valmiiksi funktiot tällaisille Hohmannin siirtoradoille, niin lasketaan käsin.

Ensin ympyrärata:

```
>>> o1 = Orbit_altitude(Earth, 300e3)
```

Sitten siirtorata;

```
>>> o2 = Orbit_elliptical(Earth,Earth.radius+300e3,Earth.radius)
```

Sitten lasketaan, paljonko on ratojen nopeuksien erotus ellipsiradan apogeumissa:

```
>>> o2.v(o2.r1)-o1.v(o2.r1)
-89.277777856096691
```

Harjoitus 8. Satelliitin radan siirtäminen alemmalta radalta ylemmälle. Esimerkkinä lento ISS:ltä Hubblelle. ISS lentää 368 km korkeudella, Hubble 573 km korkeudella. Yllätys, yllätys, tietokannasta löytyvät molemmat. Tehdään lyhennysmerkinnät (HUOM! Nämä ovat vain ratoja, massoja näille ei ole annettu) sekä valmiiksi Hohmannin rata:

```
>>> ISS = masses["ISS"].orbit
>>> Hubble = masses["Hubble"].orbit
>>> o = Orbit_elliptical(Earth,ISS.a(),Hubble.a())
```

Alemmalla radalla (ISS) poltto (delta-v) on (siirtyminen ympyäradalta ellipsiradalle):

```
>>> o.v(o.r1) - ISS.v(o.r1)
57.309705499874326
```

Kun päästään Hubblen luokse, tehdään toinen poltto (siirtyminen ellipsiradalta ympyäradalle):

```
>>> Hubble.v(o.r2) - o.v(o.r2)
56.882391591887426
```

Yksinkertaisempi keino näissä tapauksissa on käyttää suoraan ohjelmistosta löytyvää Transfer -objektia (joka vielä tulostaa vähän ylimääräistä debuggaustietoa):

```
>>> t = Transfer(ISS, Hubble)
...
Dv1 57.3097054999 Dv2 -56.8823915919
```

Vot, yksinkertaista, eikö totta?

Siirtymiseen kuluva aika on puolet ellipsiradan (siirtoradan) periodista:

```
>>> o.P()/2
2820.1627697669228
>>> o.P()/2/60
47.002712829448711
```

Harjoitus 9. Planeettojen välillä liikkuminen, (kts. kalvot alkaen s. 36).

Näissä harjoituksissa käytetään suoraan Transfer-objektia, vaikkakin se on vielä vähän puutteellinen. Kaukorannan esimerkissä on ensin laskettu delta-v Maapallon 300 kilometrin korkuiselta ympyräradalta Marsin 1000 kilometrin korkuiselle radalle (kts. kalvot s. 41):

```
>>> Mars = masses["Mars"]
>>> t = Transfer(Orbit_altitude(Earth,300e3),Orbit_altitude(Mars,1000e3))
...
Dv1 3589.98414272 Dv2 2012.12980265
```

Huomataan, että jarrutus-nopeus poikkeaa Kaukorannan laskemasta. Se johtuu siitä, että ohjelma laskee jarrutuksen paloittain planeetan vaikutusetäisyydeltä (Laplacen etäisyys) alkaen, kun taas Kaukoranta laskee äärettömästä. Tällainen muutos laskentaan oli pakko tehdä, jotta ohjelma osaisi laskea myös planeetojen kuiden välillä tapahtuvat siirtymiset.

Vastaavasti lento Maasta Venukseen:

```
>>> Venus=masses["Venus"]
>>> t = Transfer(Orbit_altitude(Earth,300e3),Orbit_altitude(Venus,1000e3))
...
Dv1 3481.65275523 Dv2 3132.88721142
```

Huomataan, että taas jarrutus on laskettu liian pieneksi (3132 m/s vs. 3186 m/s). Tätä parannellaan ehkä ohjelmiston tulevissa versioissa.