Szegedi Tudományegyetem Természettudományi és Informatikai Kar
Fizikai Intézet Optikai és Kvantumelektronikai Tanszék

SZAKDOLGOZAT

Égi mechanikai számítások Python-ban

Készítette: Pozsár Balázs, Fizika BSc szakos hallgató

Témavezető: Dr. Szalai Tamás, tudományos munkatárs

Szeged 2021

Tartalomjegyzék

Célkitűzés

Korábbi projektmunkám[1] során, egy kezdetleges, az N-test problémát szemléltető szimulációt készítettem. Ez a program a gravitációs erőképletet, valamint néhány ismert kezdeti adatot (hely, tömeg, sebesség) kiindulási alapnak használva, numerikus módszerrel kiszámítja az égitestek egymáshoz viszonyított helyzetét az idő múlásával és ezt egy animált diagramon keresztül szemlélteti is.

A szakdolgozat keretein belül, a program kódbázisát valamint a benne használt matematikai és fizikai módszerek további tárgyalását, bővítését és a szimuláció elkészítéséhez egy másik megközelítés alkalmazását tűztem ki célul.

Hasonlóképpen a korábbiakhoz, a kiindulási alap itt is a gravitációs erőképlet, de ezt már nem az N-test problémához használjuk fel, hanem a kéttest-problémához. Ennek a módszernek az az előnye, hogy nem fog függeni a szimuláció pontossága a beállított időléptéktől, vagyis attól, a már fent említett időtől ami a test egyik pontból a másikba való eljutásához szükséges. Cserébe viszont nem lehet bonyolultabb, például holdakkal és/vagy kisebb aszteroidákból illetve porból álló gyűrűvel/gyűrűrendszerrel rendelkező bolygókat tartalmazó csillagrendszerek viselkedését megvizsgálni vele. Mindezek ellenére egy nagyon szemléletes programot lehet ennek a módszernek a segítségével készíteni, amely tükrözni fogja a Kepler-törvényeket, valamint tényleges a csillagászatban használt pályaelemek használatát és kiszámítását is lehetővé teszi.

A program megírásához a Python nyelvet választottam. Elsődlegesen azért mert a fizikában, azon belül csillagászatban is egy előszeretettel alkalmazott nyelv. Könnyű vele dolgozni és hasznos, előre megírt programcsomagokat, illetve könyvtárakat lehet a munkánk során felhasználni. Ezt a dolgozatot megelőző projektmunka elkészítéséhez is ezt a nyelvet használtam, ahol ezek az állítások igazolást nyertek a számomra.

Például a vektorokkal való számítás előzetes ismereteim alapján a legtöbb programozási nyelven bonyolult tömbkezeléssel és általunk megírt programkódok segítségével, vagy nehezebben elérhető és telepíthető kiegészítő könyvtárak használatával válik csak lehetővé. Ezt Python-ban biztosítja számunkra a **NumPy**, amely szintén egy a Python-t

[1] Bolygómozgás szimulációja Python-ban

használók számára megírt kiegészítő könyvtár, de a nyelv használatához szükséges telepítési folyamatok során automatikusan a használható könyvtárak közé kerül, így nekünk már csak használnunk kell.

Bevezetés

Kéttest-probléma

A kéttest-probléma az égi mechanika egy speciális problémája, amelyben két tömegpontot vizsgálunk. Ezekre a tömegpontokra csak a kölcsönös gravitációs vonzóerők hatnak.

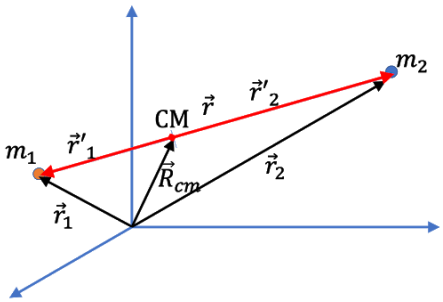
A tömegpontok mozgásegyenletei:

$$\vec{F}_{g,12} = m_1 \frac{d^2\vec{r}_1}{dt^2} = k^2 \frac{m_1 m_2}{r^3}\vec{r} \tag{1}$$

$$\vec{F}_{g,21} = m_2 \frac{d^2\vec{r}_2}{dt^2} = -k^2 \frac{m_1 m_2}{r^3}\vec{r} \tag{2}$$

ahol,

$\vec{F}_g$:	a gravitációs erővektor
k^2 :	a gravitációs állandó $\approx 6,674 \cdot 10^{-}11 \frac{\mathrm{m}^3}{\mathrm{kg} \cdot \mathrm{s}^2}$
m_1, m_2 :	a két tömegpont
$\vec{r}_1, \vec{r}_2$:	az origóból a két tömegpontba mutató helyvektor
$\vec{r}$:	m_1-ből m_2-be mutató helyvektor
r :	a tömegpontok közötti távolság



1. ábra. Kéttest-probléma ábrázolása.

physicsbootcamp.org/The-Two-Body-Problem.html, Figure 9.4.1

Adjuk össze (1)-et és (2)-t és integráljuk az idő szerint kétszer.

$$m_1\frac{d^2\vec{r}_1}{dt^2} + m_2\frac{d^2\vec{r}_2}{dt^2} = 0$$

$$m_1\frac{d\vec{r}_1}{dt} + m_2\frac{d\vec{r}_2}{dt} = \vec{a}$$

$$m_1\vec{r}_1 + m_2\vec{r}_2 = \vec{b} + \vec{a}t \tag{3}$$

Keressük meg a két tömegpontból álló rendszer tömegközéppontját (3)-as egyenlet felhasználásával.

$$\vec{R}_{cm} = \frac{\sum m_i\vec{r}_i}{\sum m_i}$$
$$\vec{R}_{cm} = \frac{m_1\vec{r}_1 + m_2\vec{r}_2}{m_1 + m_2} = \frac{\vec{b} + \vec{a}t}{m_1 + m_2}$$

Így áttérhetünk a kéttest-problémáról az egycentrum-problémára, ahol nem az Oxyz hanem a CMxyz koordináta-rendszerben vizsgáljuk a mozgásegyenleteket.

$$\vec{F}_{g,12} = m_1\frac{d^2\vec{r}_1'}{dt^2} = k^2\frac{m_1m_2}{r^3}\vec{r} \tag{4}$$
$$\vec{F}_{g,21} = m_2\frac{d^2\vec{r}_2'}{dt^2} = -k^2\frac{m_1m_2}{r^3}\vec{r} \tag{5}$$

A (4)-es és (5)-ös egyenletben szereplő $\vec{r}_1'$ és $\vec{r}_2'$ helyvektorokat meghatározhatjuk, az 1. ábra és a (3)-as egyenlet segítségével.

$$\vec{r}_1' = \vec{r_1} - \vec{R}_{cm} = \frac{\vec{b} + \vec{a}t - m_2\vec{r}_2}{m_1} - \frac{\vec{b} + \vec{a}t}{m_1 + m_2} =$$

$$= \frac{m_1\vec{b} + m_1\vec{a}t - m_1m_2\vec{r}_2 + m_2\vec{b} + m_2\vec{a}t - m_2^2\vec{r}_2 - m_1\vec{b} - m_1\vec{a}t}{m_1(m_1 + m_2)} =$$

$$= \frac{m_2(\vec{b} + \vec{a}t - \vec{r}_2(m_1 + m_2))}{m_1(m_1 + m_2)} = \frac{m_2m_1(\vec{r}_1 - \vec{r}_2)}{m_1(m_1 + m_2)} = -\frac{m_2}{m_1 + m_2}\vec{r}$$

$$\vec{r_2}' = \vec{r_2} - \vec{R}_{cm} = \frac{\vec{b} + \vec{a}t - m_1\vec{r}_1}{m_2} - \frac{\vec{b} + \vec{a}t}{m_1 + m_2} =$$

$$= \frac{m_1\vec{b} + m_1\vec{a}t - m_1^2\vec{r}_1 + m_2\vec{b} + m_2\vec{a}t - m_1m_2\vec{r}_1 - m_2\vec{b} - m_2\vec{a}t}{m_2(m_1 + m_2)} =$$

$$= \frac{m_1(\vec{b} + \vec{a}t - \vec{r}_1(m_1 + m_2))}{m_2(m_1 + m_2)} = \frac{m_1m_2(\vec{r}_2 - \vec{r}_1)}{m_2(m_1 + m_2)} = \frac{m_1}{m_1 + m_2}\vec{r}$$

Ezeket behelyettesítve (4)-be és (5)-be a következőt kapjuk, hogy

$$m_1\frac{d^2\vec{r}_1'}{dt^2} = k^2\frac{m_1m_2}{r^3}\vec{r} \rightarrow \frac{d^2\vec{r}}{dt^2} = -k^2\frac{m_1 + m_2}{r^3}\vec{r}$$

$$m_2\frac{d^2\vec{r}_2'}{dt^2} = -k^2\frac{m_1m_2}{r^3}\vec{r} \rightarrow \frac{d^2\vec{r}}{dt^2} = -k^2\frac{m_1 + m_2}{r^3}\vec{r}$$

$$\frac{d^2\vec{r}}{dt^2} = -k^2\frac{m_1 + m_2}{r^3}\vec{r} \tag{6}$$

Könnyen belátható, hogy a (6)-os összefüggés megegyezik (1) - (2)-vel.

$$m_1\frac{d^2\vec{r}_1}{dt^2} = k^2\frac{m_1m_2}{r^3}\vec{r} \rightarrow \frac{d^2\vec{r}_1}{dt^2} = k^2\frac{m_2}{r^3}\vec{r}$$

$$m_2\frac{d^2\vec{r}_2}{dt^2} = -k^2\frac{m_1m_2}{r^3}\vec{r} \rightarrow \frac{d^2\vec{r}_2}{dt^2} = -k^2\frac{m_1}{r^3}\vec{r}$$

$$\frac{d^2\vec{r}_1}{dt^2} - \frac{d^2\vec{r}_2}{dt^2} = \frac{d^2(\vec{r}_1 - \vec{r}_2)}{dt^2} = k^2\frac{m_1 + m_2}{r^3}\vec{r} \rightarrow \frac{d^2\vec{r}}{dt^2} = -k^2\frac{m_1 + m_2}{r^3}\vec{r}$$

Tehát a (6)-os egyenlet megmutatja hogyan mozog az m_2 jelzésű tömegpont az m_1 tömegpont körül.

Most mutassuk meg, hogy a mozgás síkban zajlik. Szorozzuk be a (6)-os egyenlet minkét oldalát a helyvektorral, vektoriálisan.

$$\vec{r} \times \frac{d^2\vec{r}}{dt^2} = \vec{r} \times -k^2 \frac{m_1 + m_2}{r^3} \vec{r}$$

$$\vec{r} \times \frac{d^2\vec{r}}{dt^2} = -k^2 \frac{m_1 + m_2}{r^3} (\vec{r} \times \vec{r})$$

$$\vec{r} \times \frac{d^2\vec{r}}{dt^2} = 0$$

Ha integráljuk a kapott egyenlet mindkét oldalát, a következőt kapjuk

$$\vec{r} \times \frac{d\vec{r}}{dt} = \vec{c}$$

ahol, az egyenlet bal oldala az egységnyi tömegre vonatkoztatott impulzusmomentum áll, ami eszerint konstans.

Skalárisan beszorozva az egyenlet mindkét oldalát egy vegyes szorzatot fogunk kapni,

$$\vec{r} \left(\vec{r} \times \frac{d\vec{r}}{dt} \right) = \vec{r} \cdot \vec{c}$$

amelyre alkalmazhatjuk a következő átalakítást,

$$\vec{a}(\vec{b} \times \vec{c}) = \vec{a} \cdot \vec{b} \cdot \vec{c} = \det(\vec{a}, \vec{b}, \vec{c})$$

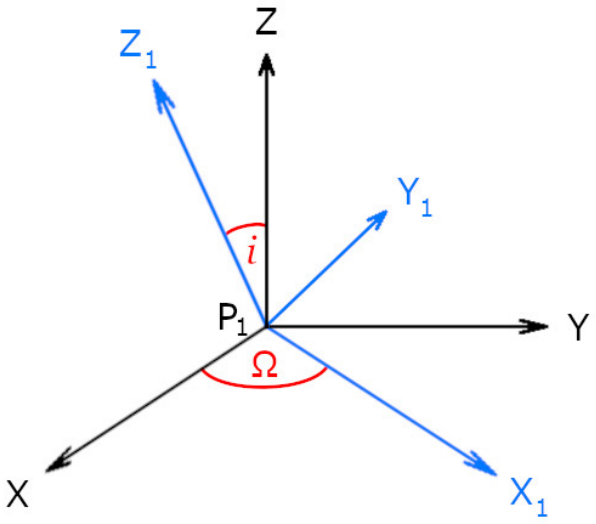
így a fent említett vegyes szorzatból azt kapjuk, hogy

$$\vec{r} \left(\vec{r} \times \frac{d\vec{r}}{dt} \right) = \begin{vmatrix} x & y & z \\ x & y & z \\ \frac{dx}{dt} & \frac{dy}{dt} & \frac{dz}{dt} \end{vmatrix} = xy\frac{dz}{dt} + yz\frac{dx}{dt} + xz\frac{dy}{dt} - yz\frac{dx}{dt} - xz\frac{dy}{dt} - xy\frac{dz}{dt} = 0$$

$$\vec{r} \cdot \vec{c} = 0$$

Ez azt jelenti, hogy $\vec{c} \neq 0$ esetén $\vec{r}$ és $\vec{c}$ egymásra merőleges, $\vec{r}$ mindig a konstans $\vec{c}$ vektorra merőleges síkban van. A mozgás tehát síkmozgás.

Legyen az alapsík a pályasík ($P_1XYZ \rightarrow P_1X_1Y_1Z_1$)!



2. ábra. Alapsík ábrázolása.

Így a mozgás egyenletei,

$$\frac{d^2x_1}{dt^2} = -k^2\frac{m_1+m_2}{r^3}x$$

$$\frac{d^2y_1}{dt^2} = -k^2\frac{m_1+m_2}{r^3}y$$

A fentebb leírtakhoz hasonlóan az impulzus momentum pedig,

$$x_1\frac{dy_1}{dt} - y_1\frac{dx_1}{dt} = c$$

A (6)-os egyenlet mindkét oldalát skalárisan beszorozva $\frac{d\vec{r}}{dt}$-vel majd idő szerint integrálva, az integrálás azonosságait felhasználva,

$$\frac{d^2\vec{r}}{dt^2}\cdot\frac{d\vec{r}}{dt} = -k^2\frac{m_1+m_2}{r^3}\vec{r}\cdot\frac{d\vec{r}}{dt} = -k^2\frac{m_1+m_2}{r^2}\hat{r}\cdot\frac{d\vec{r}}{dt}$$

$$\int\frac{d^2\vec{r}}{dt^2}\cdot\frac{d\vec{r}}{dt}\,dt = \left(\frac{dr}{dt}\right)^2 - \int\frac{d^2\vec{r}}{dt^2}\cdot\frac{d\vec{r}}{dt}\,dt \rightarrow \int\frac{d^2\vec{r}}{dt^2}\cdot\frac{d\vec{r}}{dt}\,dt = \frac{1}{2}\frac{d\vec{r}}{dt}$$

$$\begin{aligned}
\int -k^2\frac{m_1+m_2}{r^2}\hat{r}\cdot\frac{d\vec{r}}{dt}\,dt = &= -k^2(m_1+m_2)\hat{r}\int\frac{1}{r^2}\frac{d\vec{r}}{dt}\,dt = \\
&= -k^2(m_1+m_2)\hat{r}\left(\frac{\vec{r}}{r^2}-\int -2\frac{\vec{r}}{r^3}\,dt\right) = \\
&= -k^2(m_1+m_2)\hat{r}\left(\frac{\hat{r}}{r}-\frac{2\hat{r}}{r}+\hat{r}h\right) = \\
&= k^2\frac{m_1+m_2}{r}|\hat{r}|^2+h = \\
&= k^2\frac{m_1+m_2}{r}+h \\
\frac{1}{2}\frac{d\vec{r}}{dt} &= k^2\frac{m_1+m_2}{r}+h \qquad (7)
\end{aligned}$$

Amit átrendezve az energia állandóságára vezető összefüggést kapunk

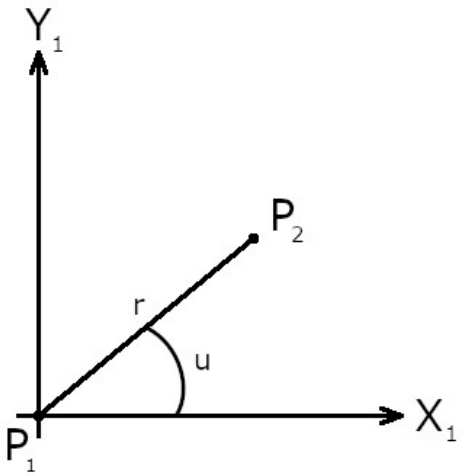
$$\frac{1}{2}\frac{d\vec{r}}{dt}-k^2\frac{m_1+m_2}{r}=h \qquad (8)$$

ahol h konstans, a rendszer összenergiája.

Felhasználva, hogy $\mu = k^2(m_1+m_2)$

$$v^2=\frac{2\mu}{r}+2h \qquad (9)$$

Térjünk át polárkoordinátákra és határozzuk meg a mozgás pályáját.



3. ábra. Áttérés polárkoordinátákra.

$$x_1 = r\cdot\cos u$$

$$y_1 = r\cdot\sin u$$

A mozgásegyenletek így,

$$\frac{dx_1}{dt} = \frac{dr}{dt}\cos u - r \cdot \sin u \frac{du}{dt}$$

$$\frac{dy_1}{dt} = \frac{dr}{dt}\sin u + r \cdot \cos u \frac{du}{dt}$$

az impulzusmomentum pedig

$$c = x_1\frac{y_1}{dt} - y_1\frac{x_1}{dt} = r \cdot \cos u \left(\frac{dr}{dt}\cos u - r \cdot \sin u \frac{du}{dt}\right) - r \cdot \sin u \left(\frac{dr}{dt}\sin u + r \cdot \cos u \frac{du}{dt}\right)$$

$$c = r^2 \frac{du}{dt} \tag{10}$$

A (9)-es egyenletbe behelyettesítve v-t, ahol így

$$v^2 = |\vec{v}|^2 = \left(\frac{dx_1}{dt}\right)^2 + \left(\frac{dy_1}{dt}\right)^2 = \left(\frac{dr}{dt}\right)^2 \cos^2 u + r^2 \sin^2 u \left(\frac{du}{dt}\right)^2$$

$$-2r\frac{dr}{dt}\frac{du}{dt}\sin u \cdot \cos u + 2r\frac{dr}{dt}\frac{du}{dt}\sin u \cdot \cos u$$

$$+ \left(\frac{dr}{dt}\right)^2 \sin^2 u + r^2 \cos^2 u \left(\frac{du}{dt}\right)^2$$

$$v^2 = \left(\frac{dr}{dt}\right)^2 + r^2\left(\frac{du}{dt}\right)^2 = \frac{2\mu}{r} + 2h \tag{11}$$

A mozgás pályájának meghatározásához, a fenti egyenletben előbb meg kell határozni $\frac{du}{dt}$-t a (10)-es egyenletből.

$$c = r^2\frac{du}{dt} \rightarrow \frac{du}{dt} = \frac{c}{r^2}$$

Ennek felhasználásával $\frac{dr}{dt}$-nél át kell térni u szerinti deriválásra.

$$\frac{dr}{dt} = \frac{dr}{du}\frac{du}{dt} = \frac{dr}{du}\frac{c}{r^2} = -\frac{d}{du}\left(\frac{c}{r}\right)$$

Ezeket pedig behelyettesítve (11)-be kapjuk, hogy

$$\left[-\frac{d}{du}\left(\frac{c}{r}\right)\right]^2 + \frac{c^2}{r^2} = \frac{2\mu}{r} + 2h$$

$$-\frac{d}{du}\left(\frac{c}{r}\right) = \sqrt{\frac{2\mu}{r} + 2h - \frac{c^2}{r^2}} = \sqrt{2h + \frac{\mu^2}{c^2} - \left(\frac{c}{r} - \frac{\mu}{c}\right)^2}$$

mivel $\frac{\mu}{c} = konstans$, ezért írhatjuk, hogy

$$-\frac{d}{du}\left(\frac{c}{r} - \frac{\mu}{c}\right) = \sqrt{2h + \frac{\mu^2}{c^2} - \left(\frac{c}{r} - \frac{\mu}{c}\right)^2}$$

Itt bevezetünk két változót

$$R = \frac{c}{r} - \frac{\mu}{c}, K = \sqrt{2h + \frac{\mu^2}{c^2}}$$

Tehát

$$-\frac{dR}{du} = \sqrt{K^2 - R^2}$$

Így ez az egyenlet a változók szétválasztásával integrálható

$$-\int \frac{1}{K\sqrt{1 - \frac{R^2}{K^2}}} dR = \int du$$

Elvégezve az integrálást az egyenlet mindkét oldalán,

$$\cos^{-1}\frac{R}{K} = u - \omega$$

ahol ω egy integrációs állandó (a pericentrum argumentuma nevű szögmennyiség). Ezt átrendezve

$$R = K\cos(u - \omega)$$

Visszahelyettesítve R-t és K-t az egyenletbe, majd azt átalakítva,

$$\frac{c}{r} - \frac{\mu}{c} = K\cos(u - \omega) = \sqrt{2h + \frac{\mu^2}{c^2}}\cos(u - \omega)$$

$$\frac{c}{r} = \frac{\mu}{c}\left[1 + \sqrt{1 + 2h + \frac{c^2}{\mu^2}}\cos(u - \omega)\right]$$

Ebből r pedig már könnyedén kifejezhető.

$$r = \frac{\frac{c^2}{\mu}}{1 + \sqrt{1 + 2h + \frac{c^2}{\mu^2}}\cos(u - \omega)}$$

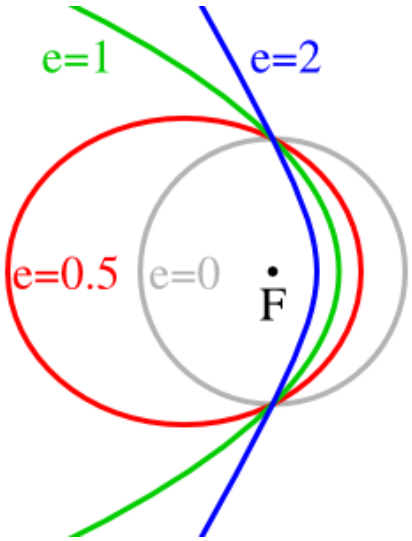
Bevezetve az alábbiakat, a pálya egyenlete

$$
\begin{aligned}
p &= \frac{c^2}{\mu}: && \text{pericentrum-távolság}\\
e &= \sqrt{1+2h+\frac{c^2}{\mu^2}}: && \text{excentricitás}\\
\nu &= u-\omega: && \text{valódi anomália}
\end{aligned}
$$

$$
r = \frac{p}{1+e\cos\nu} \tag{12}
$$

Ez egy kúpszelet egyenlete, amiből a pálya alakja lehet

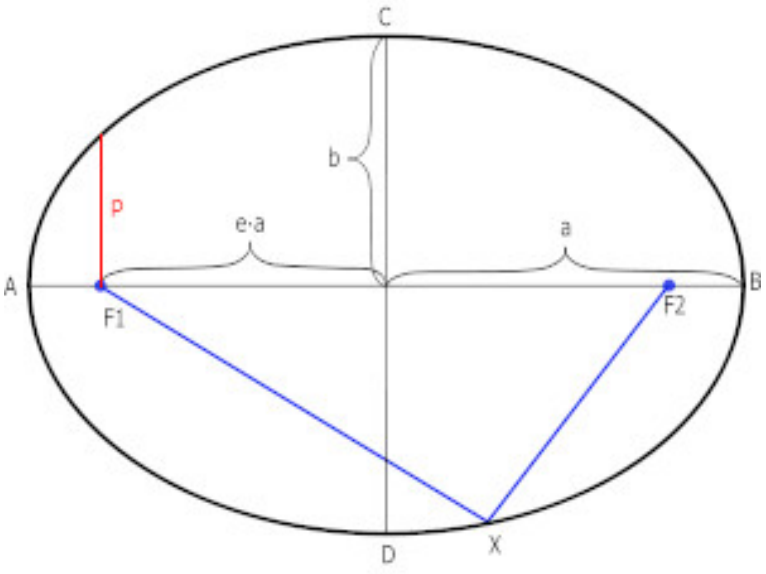
1. Ellipszis $e<1$, $h<0$ (a teljes mechanikai energia negatív)
2. Parabola $e=1$, $h=0$ (a teljes mechanikai energia nulla)
3. Hiperbola $e<1$, $h<0$ (a teljes mechanikai energia pozitív)



4. ábra. Kúpszeletek ábrázolása.
kids.kiddle.co/Orbital_eccentricity

A p paraméter vagy pericentrum-távolság, a kúpszelet nagytengelyére az F1 fókuszban állított merőleges szakasz, melynek másik végpontja a kúpszeleten van.

A ν valódi anomáliát az r rádiusznak a kúpszelet nagytengelyével bezárt szögét, a pericentrumtól (az alábbi ábrán az A pont) kiindulva az óramutató járásával ellenkező irányban mérjük.



5. ábra. Ellipszis ábra.

hu.wikipedia.org/wiki/Ellipszis

Az r távolság akkor a legkisebb ha a ν valódi anomália nulla ($u = \omega$), azaz

$$r_{min} = r(\nu = 0^\circ) = \frac{p}{1+e}$$

Ellipszis alakú pálya esetén r távolságnak lehet maximális értéke a r($\nu = 180^\circ$)-ban, ez a távolság az apocentrum.

$$r_{max} = r(\nu = 180^\circ) = \frac{p}{1-e}$$

Ellipszis-pálya esetén:

$$p = a(1-e^2)$$

ahol a az ellipszis félnagytengelye, így a pályaegyenlet felírható a következőképpen,

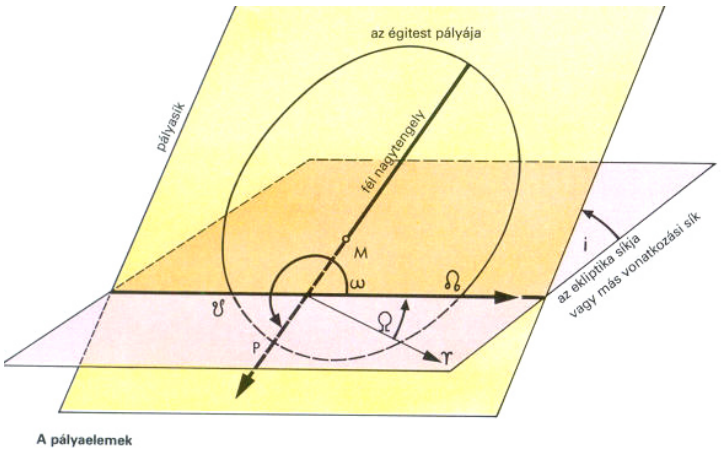
$$r = \frac{a(1-e^2)}{1+e\cos\nu} \tag{13}$$

Pályaelemek

A pályaelemek egy égitest gravitációs terében keringő objektumok (pl. kettőscsillagok komponensei, bolygók, holdak, stb.) pályájának pontos meghatározására szolgáló paraméterek.

Általában az alábbi hat paraméterrel adják meg egy keringő objektum pályáját:

i:	pályahajlás vagy inklináció	a keringési sík hajlásszöge az alapsíkhoz képest
a:	a pályaellipszis félnagytengelye	
e:	numerikus excentricitás	a pályaellipszis lapultságát adja meg, definíció szerint: $e = \frac{\sqrt{a^2 - b^2}}{a}$, ahol b a pályaellipszis fél kistengelye
ω:	a pericentrum (alábbi ábra P pont) távolsága a felszálló csomótól	a keringési síkban mérjük, a pericentrum iránya és a felszálló csomó által bezárt (pozitív irányban felvett) szög nagysága
Ω:	a felszálló csomó hossza	az alapsíkban, az alapirány és a felszálló csomó által bezárt szög nagysága
τ:	a keringő égitest (valamelyik) pericentrum-átmenetének időpontja	



6. ábra. Pályaelemek ábra.

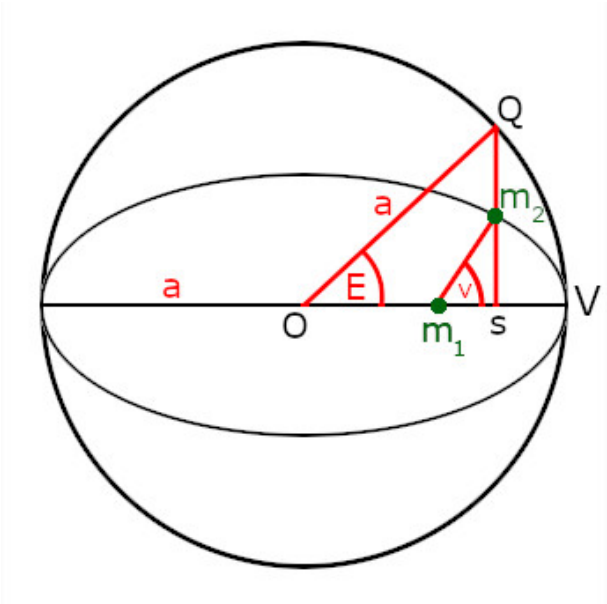
astro.u-szeged.hu/oktatas/csillagaszat/5_Egi_mechanika/egi_mechanika.htm

A mozgás időbeli lefolyása

A mozgás pályájának ismeretében meghatározható a mozgás időbeli lefolyása, azaz az $r(t)$ függvény. Mivel r-nek a ν valódi anomáliától való függését ismerjük, a feladat a $\nu(t)$ függvény meghatározása. Írjuk fel az impulzusmomentum-egyenletet v felhasználásával:

$$r^2 \frac{d\nu}{dt} = k\sqrt{(m_1 + m_2) \cdot a(1 - e^2)} \tag{14}$$

Ez az egyenlet azonban véges formában nem integrálható, így szükség van ν helyett egy új változó bevezetésére. Láttuk, hogy a mozgás pályája ellipszis, parabola vagy hiperbola, az energiától függően. Az új változó bevezetése függ a pálya típusától. Mivel a gyakorlatban elliptikus pályák fordulnak elő leggyakrabban (pl. bolygók mozgása), itt csak az elliptikus mozgás esetét tárgyaljuk.



7. ábra. Pályaellipszis ábrázolása.

Elliptikus mozgás esetén a ν valódi anomália helyett vezessük be az E excentrikus anomáliát. Rajzoljuk meg ehhez az ellipszis főkörét, ami egy a sugarú kör az ellipszis középpontja körül. Az m_2 tömegponton át húzzunk egyenest az ellipszis nagytengelyére merőleges irányban, így kapunk egy metszéspontot a főkörön. Ezt a metszéspontot az ellipszis középpontjával összekötve a kapott szög az E excentrikus anomália.

Határozzuk meg az összefüggést ν és E között. Ehhez szükségünk lesz $\overline{\mathrm{m_1s}}$ és $\overline{\mathrm{sm_2}}$ szakaszokra.

Vizsgáljuk meg a kapcsolatot a kör és az ellipszis egyenletei között.

Az ellipszis egyenlete:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \rightarrow y_e = \frac{b}{a}\sqrt{a^2 - x^2}$$

A kör egyenlete:

$$a = b$$
$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \rightarrow y_k = \sqrt{a^2 - x^2}$$

Ezekből pedig,

$$\frac{y_e}{y_k} = \frac{b}{a}$$

Írjuk fel az egyes szakaszokat,

$$\begin{aligned}
\overline{\mathrm{m_1m_2}} &= r \\
\overline{\mathrm{Om_1}} &= ae \\
\overline{\mathrm{sQ}} &= a\sin E \\
\overline{\mathrm{sm_2}} &= \frac{b}{a}a\sin E = b\sin E = a\sqrt{1-e^2}\sin E = r\sin\nu \\
\overline{\mathrm{m_1s}} &= a\cos(E) - ae = a(\cos(E) - e) = r\cos\nu
\end{aligned}$$

Emeljük négyzetre és adjuk össze $r\sin\nu$-t és $r\cos\nu$-t.

$$\begin{aligned}
r^2 &= a^2(\cos^2 E - 2e\cos E + e^2 + \sin^2 E - e^2\sin^2 E) = \\
&= a^2(1 + e^2 - 2e\cos E - e^2\sin^2 E) = \\
&= a^2(1 - 2e\cos E + e^2(1 - \sin^2 E)) = \\
&= a^2(1 - 2e\cos E + e^2\cos^2 E) = \\
&= a^2(1 - e\cos E)^2
\end{aligned}$$

$$r = a(1 - e\cos E)$$

Kivonva r-ből $r\cos\nu$-t,

$$r - r\cos\nu = r(1-\cos\nu) = a(1 - e\cos E - \cos E + e) = a((1+e)(1-\cos E))$$

Hozzáadva r-hez $r\cos\nu$-t,

$$r + r\cos\nu = r(1+\cos\nu) = a(1 - e - e\cos E + \cos E) = a((1-e)(1+\cos E))$$

Ez utóbbi két egyenlet hányadosát véve,

$$\frac{1-\cos\nu}{1+\cos\nu} = \frac{(1+e)(1-\cos E)}{(1-e)(1+\cos E)}$$

Itt pedig az egyenlet mindkét oldalának gyökvonása után felhasználhatjuk a következő trigonometrikus azonosságot,

$$\tan\left(\frac{x}{2}\right) = \pm\sqrt{\frac{1-\cos x}{1+\cos x}}$$

A fentiek alapján, kiderül, hogy v és E kapcsolata csak e-től függ:

$$\tan\frac{\nu}{2} = \sqrt{\frac{1+e}{1-e}}\tan\frac{E}{2} \tag{15}$$

Az r helyvektor nagysága pedig így adható meg:

$$r = a(1 - e\cos E) \tag{16}$$

Beírva E-t ν helyére, az impulzusmomentum-egyenlet így alakul:

$$(1 - e\cos E)\frac{dE}{dt} = a^{-3/2}k\sqrt{m_1 + m_2} \tag{17}$$

Az egyenletet integrálva megoldásként az ún. Kepler-egyenletet kapjuk:

$$E - e\sin E = n(t - \tau) = M \tag{18}$$

ahol M a középanomália, τ a pericentrum-átmenet időpontja, n pedig az ún. középmozgás:

$$n = \frac{2\pi}{T} = \mu^{1/2}a^{-3/2} \tag{19}$$

A transzcendens Kepler-egyenletből $E = E(t)$ meghatározható, de csak egy végtelen sor formájában (numerikus módon megoldható).

Megvalósítás

A program fő célja, hogy a valósághoz közelálló szimulációt mutasson be a csillagászatban is használatos pályaelemek használatával és kiszámításával. Ehhez a fent leírt számításokat kell tudnia elvégeznie, méghozzá másodpercenként többször, a program elindításától kezdve egészen addig amíg fut. Mivel nem csak egy kettős rendszert szeretnénk szimulálni, habár az is elég lenne a program helyes működésének vizuális igazolásához, lehetségessé kell tenni, hogy több égitest mozgását is nyomon követhessük a futási idő alatt. Ezt úgy lehet megvalósítani ha több égitestre is elvégezzük a számításokat egy iteráció alatt.

A szimulációhoz az alábbi csomagokra lesz szükségünk.

pyqtgraph: Egy vizualizációs programcsomag ami egy fejlettebb animációt teszi lehetővé, mint egy szimpla animált diagram. Szükség lesz belőle az **opengl Qt**, ezen belül pedig a **QtCore** és **QtGui** alkönyvtárakra.

numpy: Egy tömbkezelésre felhasználható könyvtár, amellyel gyorsan és könnyen végezhetünk műveleteket egy, két, vagy akár egyszerre több adattömbbel is, illetve tartalmazza a számunkra szükséges matematikai konstansokat mint például a π és a trigonometrikus függvényeket.

pandas: Egy adatkezelésre szolgáló könyvtár. Segítségével könnyedén feldolgozhatunk kis és nagy mennyiségű adatot is, ilyen lesz például a bemeneti adathalmazunk ami tartalmazza az égitestek adatait.

datetime: Az idő kezeléshez szükséges könyvtár.

A szimulációt és a hozzá szükséges műveleteket tartalmazó függvényeket egy osztályban definiáljuk. Ahhoz, hogy a későbbiekben könnyedén tudjunk új égitestet hozzáadni a rendszerhez, érdemes egy jól strukturált adathalmazt létrehozni, amit egyszerű átlátni és tartalmazza a szükséges adatokat.

A fent említett osztály meghívásakor ezt a ***dictionary*** típusú adathalmazt adjuk meg egyedüli paraméterként, amely a következő adatokat tartalmazza:

m:	tömeg	[kg]
d:	átmérő	[m]
a:	a pályaellipszis félnagytengelye	[m]
e:	numerikus excentricitás	
i:	pályahajlás vagy inklináció	[°]
τ:	a pericentrum-átmenet időpontja	[s]
r:	"aktuális" helyvektor	[m]
r_o:	"előző" helyvektor	[m]
v:	valódi anomália	[rad]
E:	excentrikus anomália	[rad]
M:	középanomália	[rad]
color:	az égitest színe RGB-ben	np.array([r,g,b])

Ennek az osztálynak szüksége van négy olyan függvényre ami az animáció futásáért és a hozzá szükséges adatok kezeléséért felelős.

animation:	Meghívja a **start** függvényt és beállít egy timer-t ami adott időközönként meghívja az **update** függvényt.
start:	Elindítja az animáció kezelőfelületét.
update:	Meghívja a fizikai paraméterek kiszámításához szükséges függvényeket, majd meghívja a **setPositions** függvényt.
setPositions:	Végig iterál az égitesteken és az animáción beállítja a hozzájuk tartozó objektumok koordinátáit.

Ezek után ahhoz, hogy kiszámítsuk az égitestek pozícióit a fentiek alapján, 4 lépésen kell végig menni.

1. Középanomália kiszámítása
2. A közép anomáliából (Newton-módszer segítségével) az excentrikus anomália meghatározása

3. Az excentrikus anomáliából meghatározható a valódi anomália

4. Pályaegyenlet megoldása és átváltás Descartes-koordinátákra

Megjegyezzük, hogy a központi objektum esetében ezek a számítások nem szükségesek, mivel azt feltételezzük, hogy az nem mozdul a szimuláció során.

Középanomália kiszámítása

Ha a (18)-as egyenletből n-t behelyettesítjük (19)-be,

$$M = \sqrt{\frac{\mu}{a^3}}(t - \tau)$$

Így csak $(t - \tau)$-t kell meghatározni, ami a két pozíció váltás között eltelt idő. Egyelőre fixáljuk le τ-t és mondjuk azt, hogy megegyezik a program elindításának idejével amennyiben nincs beállítva neki érték. Később megadhatjuk az egyes objektumok τ paraméterének értékét így pl. a Naprendszer esetében látni fogjuk a bolygók aktuális elhelyezkedését a Naphoz képest.

A függvény neve ***calcMeanAnomaly*** lesz. Ebben a függvényben végig iterálunk az objektumokon és kiszámítjuk, majd beállítjuk a hozzájuk tartozó **közép anomáliát**.

Állítsunk be egy változót az osztályunkban (**T_CURRENT**), amiben az aktuális időt tároljuk el, ez fog megfelelni t-nek. Az esetleges program fordulási hibák elkerülésére ellenőrizzük le, hogy van-e már értéke ennek a változónak, ha nincs akkor állítsuk be a program elindításának idejét a ***datetime.datetime.now()*** függvénnyel.

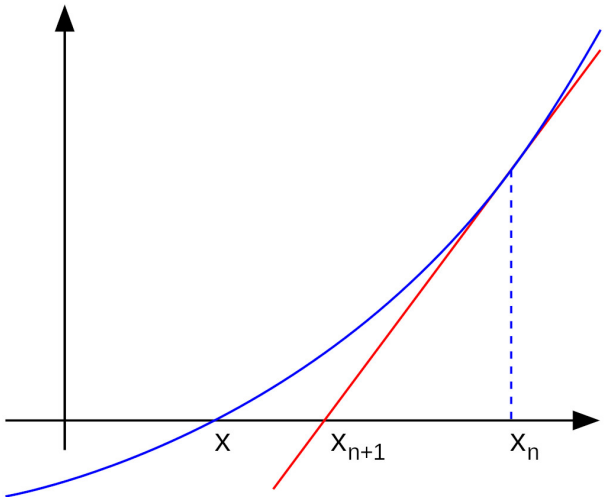
Szintén le kell ellenőrizni, hogy a **középanomália** kiszámításához szükséges adatok be vannak-e állítva, majd a fenti képletet használva kiszámíttatjuk a programmal a **közép anomáliát** és azt hozzá is adjuk/frissítjük az aktuális égitest adataihoz.

```
def calcMeanAnomaly(self):
    if(self.T_CURRENT is None):
```

```
            self.T_CURRENT = datetime.datetime.now()
        self.T_CURRENT += datetime.timedelta(days=1)

        for obj in self.objects:
            if obj == 'SUN':
            continue

            if(self.objects[obj].tau is None):
                self.objects[obj].tau = datetime.datetime.now()

            if self.objects[obj].a is None or self.objects[obj].m is None:
                print("Value missing from object ", obj)
                print("Semi-major axis: ", self.objects[obj].a)
                print("Object's mass: ", self.objects[obj].m)
                return

            mu = G*(self.objects['SUN'].m+self.objects[obj].m)
            M = np.sqrt(
                mu/self.objects[obj].a**3)*(
                    (self.T_CURRENT-self.objects[obj].tau).total_seconds()
                )
            self.objects[obj].M = M
        return
```

Excentrikus anomália kiszámítása

Mivel az excentrikus anomáliát csak numerikus megközelítéssel lehet meghatározni, használnunk kell ehhez a ***Newton-módszert***. Ez a módszer arra használatos, hogy a valós függvények gyökeit (zérushelyeit) megközelíthessük. Ehhez szükségünk van egy kiindulópontra, ami az igazi gyökhöz elég közel áll, így gyorsan megtalálhatjuk a keresett értéket. Ez a gyök a megadott kiindulási pontban megközelítőleg az ehhez a ponthoz húzott érintőn (a függvény deriváltja) található. Ennek az érintőnek a metszéspontja az x-tengellyel egy jobb közelítést ad a függvény gyökéről mint a kezdeti pontunk.



8. ábra. Newton-módszer ábra.

hu.wikipedia.org/wiki/Newton-módszer

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Ez a folyamat iterálható, tehát például egy ciklusba rendezve felhasználhatjuk jó közelítéssel az excentrikus anomália meghatározására.

Hozzunk létre egy **NewtonMethod** nevű függvényt az osztályon kívül. Öt változót kell ennek a függvénynek átadni, egy kezdeti értéket p_0; egy tolerancia határt ami alatt elfogadjuk a kapott értéket **tol**; egy maximális iterációszámot **N**, hogy elkerüljük a végtelen ciklus lehetőségét; a függvényt **f** és a függvény deriváltját **df** aminek a gyökét szeretnénk megtalálni.

A függvényen belül beállítunk egy ciklusszámlálót, ami ha eléri a maximális iterációszámot megakadályozza a ciklus újbóli lefutását, így a függvény visszatérhet hibaüzenettel. Belépünk egy ciklusba a meghatározott előfeltételekkel, végrehajtjuk a fentebb leírt műveletet, majd megnézzük, hogy a kapott gyök és a kezdeti érték különbségének abszolút értéke kisebb-e mint a megadott tolerancia határ. Ha kisebb, akkor visszatérünk a kapott gyökkel, ha nagyobb akkor megnöveljük egyel a ciklusszámlálót és az új kezdeti értéknek beállítjuk az aktuális iterációban kapott gyököt.

```
def NewtonMethod(p_0, tol, N, f, df):
    index = 1
    while index <= N:
        p = p_0 - (f(p_0)/df(p_0))
        if np.abs(p-p_0) < tol:
            return p
        index += 1
        p_0 = p

    print("Cannot determine requested value...")
    return False
```

Az excentrikus anomália kiszámításáért a **calcExccentricAnomaly** függvény fog felelni. A függvény végig iterál az égitesteken, ellenőrzi, hogy a szükséges adatok be vannake állítva, ha nem akkor vagy visszatér vagy beállít egy kezdeti értéket.

A (18)-as egyenletből határozzuk meg a függvényt és annak deriváltját amit tovább kívánunk adni a **NewtonMethod** függvénynek.

Rendezzük át a (18)-as egyenletet,

$$f(x) = M - E + e \sin E = 0$$

majd deriváljuk le E szerint

$$\frac{df(x)}{dE} = -1 + e \cos E$$

Ezeket a függvényeket fogjuk továbbadni, hogy meghatározhassuk az excentrikus anomáliát, továbbiakban E-t. Állítsuk be a többi paramétert is, a kezdeti értéknek a **közép anomáliát** adjuk meg arra számítva, hogy ez elég közeli érték a tényleges E értékéhez. Mivel ez a közelítési folyamat elég gyors, így a maximális iteráció számnak adjunk 10-et. Tolerancia határnak pedig állítsunk be 10^{-5}-es nagyságrendet. Ennek a következő a magyarázata:

Az egyszerűség kedvéért vegyük alapul a Föld bolygót és tegyük fel, hogy tökéletes körpályán ($e = 0$) kering az (x,y) síkban a Nap körül és a $t = 0$-ban az (x,0) pontban

helyezkedik el. Ebben az esetben egy nap eltelte után a Föld excentrikus anomáliája

$$E = M = \frac{2\pi}{T} \cdot 86400\text{s} \pm tol = \frac{2\pi}{365} \pm 10^{-5}[\text{rad}]$$

Mekkora időbeli különbséget is jelent az a 10^{-5}[rad] eltérés? Könnyedén meghatározhatjuk.

$$2\pi[\text{rad}] = 365\text{nap} \rightarrow 1[\text{rad}] = \frac{365}{2\pi}\text{nap}$$
$$10^{-5}[\text{rad}] = \frac{365}{2\pi}10^{-5}\text{nap} \approx 5.8 \cdot 10^{-4}\text{nap} \approx 50s$$

Tehát nagyjából percre pontosan meghatározhatjuk ekkora közelítéssel a Föld helyzetét a Nap körüli pályán, ezért ez a közelítés elég jónak számít. Ez az eltérés a távolság növekedésével természetesen nő, ez a Neptunusz esetében már 2,3 órát jelentene, de annak 165 éves keringési periódusával még így is azt mondhatjuk, hogy jó a közelítés.

Ezek alapján adjuk meg a fenti paramétereket a **NewtonMethod** függvénynek és állítsuk be az általa megadott értéket mint az égitest excentrikus anomáliáját.

```
def calcEccentricAnomaly(self):
    for obj in self.objects:
        if obj == 'SUN':
            continue

        if self.objects[obj].M is None or self.objects[obj].e is None:
            print("Values missing from object ", obj)
            print("Eccentricity: ", self.objects[obj].e)
            print("Mean Anomaly: ", self.objects[obj].M)
            return

        if self.objects[obj].E is None:
            self.objects[obj].E = 0.0

        def f(E, M=self.objects[obj].M, e=self.objects[obj].e):
            return M - E + e*np.sin(E)
```

```
        def df(E, e=self.objects[obj].e):
            return - 1 + e*np.cos(E)

        p_0 = self.objects[obj].M
        N = 10
        tol = 1e-5
        E = NewtonMethod(p_0, tol, N, f, df)
        self.objects[obj].E = E
    return
```

Valódi anomália kiszámítása

Ennek a paraméternek a kiszámításáért a **calcRealAnomaly** nevű függvény fog felelni. Ez a függvény is égitestek listáján iterál végig, megnézve hogy a valódi anomália kiszámításához szükséges egyéb (e, E) paraméterek be vannak-e állítva.

Ezek után a (15)-ös egyenletet felhasználva,

$$\tan\frac{v}{2} = \sqrt{\frac{1+e}{1-e}}\tan\frac{E}{2} \rightarrow v = 2\arctan\left(\sqrt{\frac{1+e}{1-e}}\tan\frac{E}{2}\right)$$

beállítja a valódi anomália értékét az adott égitestnek.

```
def calcRealAnomaly(self):
    for obj in self.objects:
        if obj == 'SUN':
            continue

        if self.objects[obj].e is None or self.objects[obj].E is None:
            print("Values missing from object ", obj)
            print("Eccentricity: ", self.objects[obj].e)
            print("Eccentric Anomaly: ", self.objects[obj].E)
            return

        e = self.objects[obj].e
```

```
            E = self.objects[obj].E
            v = 2 * np.arctan(np.sqrt((1+e)/(1-e))*np.tan(E/2))
            self.objects[obj].v = v
        return
```

Pályaegyenlet megoldása és átváltás Descartes-koordinátákra

A pályaegyenlet megoldását és a koordináták átváltását a **calcPosition** függvény fogja elvégezni. Ahogyan a többi függvény ez is végig fut az égitesteken, leellenőrzi, hogy hiányoznak-e a számításokhoz szükséges adatok, ha igen akkor beállít egy kezdőértéket. A (13)-as egyenletbe behelyettesítve kiszámítja a rádiusz vektort, ami polárkoordinátákkal van megadva és abból kiszámítja a Descartes-koordinátákat, végül pedig eltárolja az előző koordinátákat és frissíti az újakat. Ez utóbbi művelet az animáció sajátosságai miatt szükséges, mivel a két koordináta adat különbsége fogja megmondani a programnak, hogy mennyivel kell eltolni a koordinátákat az előzőekhez képest.

```
def calcPosition(self):
    for obj in self.objects:
        if(self.objects[obj].v is None):
            self.objects[obj].v = 0.0

        if(self.objects[obj].M is None):
            self.objects[obj].M = 0.0

        if(self.objects[obj].i is None):
            self.objects[obj].i = 0.0

        if(self.objects[obj].e is None):
            self.objects[obj].e = 0.0

        a = self.objects[obj].a
        e = self.objects[obj].e
        v = self.objects[obj].v
```

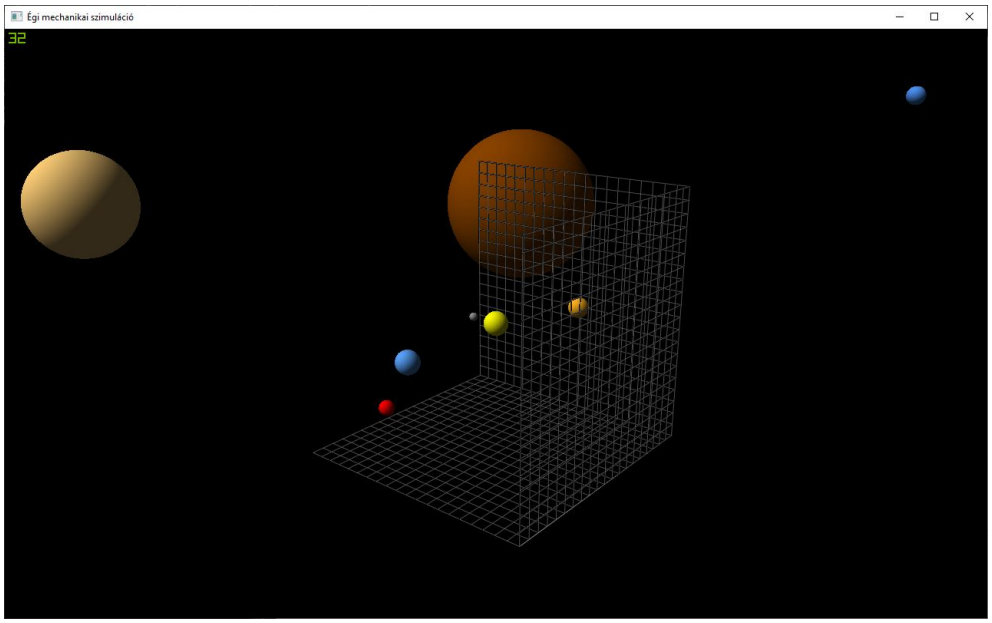
```
            r = (a*(1-e**2)) / (1+e*np.cos(v))

            x = r*np.cos(v)
            y = r*np.sin(v)

            self.objects[obj].r_o = self.objects[obj].r
            self.objects[obj].r = 10*np.array([x, y, 0])
        return
```

Végezetül pedig be kell állítani az osztály konstruktorában, az animáció ablakának megnyitásáért felelős paramétereket, ki kell nyerni a megadott adathalmazból az adatokat számunkra megfelelő formátumban.

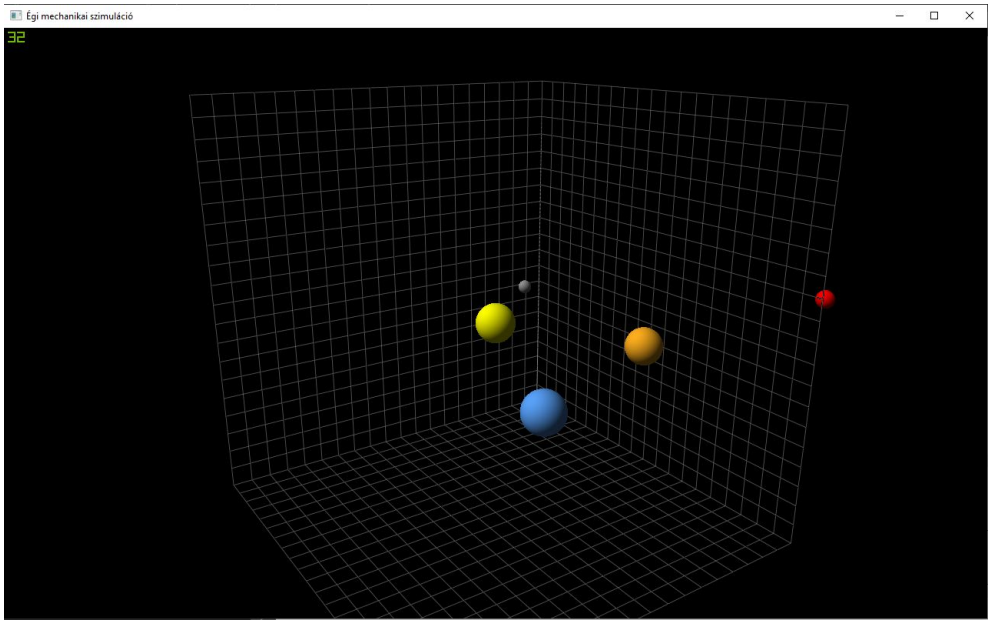
Az animációhoz hozzáadunk egy koordináta hálót is, ahol tíz beosztás lesz egy csillagászati egység. Ki kell számítani a kezdeti adatokat és beállítani a bolygóknak megfelelő gömbalakzatokat (távolság, méret szín, stb.), figyeljünk az égitestek méretének és a középponti objektumtól való távolság helyes beállítására, mert könnyedén előfordulhat az a helyzet, hogy a központi objektum mérete túllóg az első pár körülötte keringő égitest pályáján és így elnyeli őket.

Az animációs programcsomag biztosít egy forgatásra alkalmas függvényt minden hozzátartozó elem számára, amelyet használhatunk a pályahajlás vagy inklináció és a pericentrum argumentumának beállítására.

Állítsuk be még az animáció lépésközét, lehetőleg akkorára, ami folytonos mozgás látszatát kelti és nem okozza az égitestek "ugrálását". Ezek után meghívhatjuk a program elindításakor az osztály **animation** függvényét, amellyel elindul az szimuláció, amelynek az egy-egy képkockáját az alábbi két kép mutatja.



9. ábra. Szimuláció futtatása 1. kép



10. ábra. Szimuláció futtatása 2. kép

Az első képen a Neptunusz bolygó és a Plútó törpebolygó kivételével, a rendszer mindegyik tagja látszik. Ez utóbbit egy kicsit keresni kell az animáción mivel nagyon pici, nem véletlenül lett eltávolítva a bolygók osztályából. A második képen pedig egy közelebbi nézőpontból figyelhetjük meg a belső bolygók mozgását.

A bolygók és a Nap egymáshoz mért távolságát, illetve méretét még lehetne változtatni, hogy a Nap mérete a bolygókéhoz képest a valóságot jobban tükrözze, de az nagyon sok

"zoom"-olással és kereséssel járna. Ez egy olyan probléma amit nem lehetett megoldani, de a legtöbb hasonló szimuláció nem is igazán foglalkozik vele.

Összegzés

Ez a program alkalmas az égitesteknek a központi objektumhoz viszonyított mozgását bemutatni egy szemléletes animációt adva végeredményül, a pályaelemek és a tömegek ismeretében. A pontosság a már korábban megírt programhoz képest valóban nem függ a beállított időléptéktől. Az égitestek mozgása a központi objektum körül pontosnak mondható, feltételezve, hogy a szimulált rendszerben és annak közvetlen környezetében semmi "komolyabb" változás nem történik, például nagyobb tömegű objektum elhaladása a rendszer mellett, hiszen az égitestek gravitációs mezejének egymásra gyakorolt hatását nem tudjuk megfigyelni a szimuláció során.

Ebből adódóan kettős rendszereket, központi csillag - bolygó, bolygó - hold/műhold, tökéletesen megtudunk valósítani a programmal és ahogy azt a dolgozat elején említettem, aszteroidaövet egy csillag körül vagy gyűrűvel/gyűrűrendszerrel rendelkező bolygókat nem tudunk a szimulációhoz adni, csak ha azok minden egyes elemének pályaelemeit ismerjük. Ez viszont hatalmas mennyiségű adat bevitelével jár, ami ha a rendelkezésünkre áll, akkor a kódbázis minimalizálásának érdekében érdemes egy külön fájlban megadni és beolvasni azt.

Köszönetnyilvánítás

Szeretnék ezúton is köszönetet mondani témavezetőmnek, Dr. Szalai Tamásnak a sok javaslatért, segítségért, valamint a folyamatos lektorálásért és türelemért.

Források

- Csillagászat elektronikus tananyag,
  http://astro.u-szeged.hu/oktatas/csillagaszat.html
- Érdi Bálint - Égi mechanika 1996.,
  https://mek.oszk.hu/04800/04800/04800.pdf

Függelék

A program forráskódja

```
import pyqtgraph as pg
import pyqtgraph.opengl as gl
from pyqtgraph.Qt import QtCore, QtGui
import numpy as np
import pandas as pd
import datetime
import sys

G = 6.674e-11  # [m**3/kg*s**2]
AU = 1.5e11  # [m]

# NewtonMethod(p_0, tol, N, f, df)
def NewtonMethod(p_0, tol, N, f, df):
    index = 1
    while index <= N:
        p = p_0 - (f(p_0)/df(p_0))
        if np.abs(p-p_0) < tol:
            return p
        index += 1
        p_0 = p

    print("Cannot determine requested value...")
    return False

class TwoBodyOrbits:
    def __init__(self, objectData):
        self.objects = pd.DataFrame.from_dict(objectData)
        self.traces = dict()
```

```
self.T_CURRENT = None

self.app = QtGui.QApplication(sys.argv)
self.w = gl.GLViewWidget()
self.w.opts['distance'] = 50
self.w.setWindowTitle('Égi mechanikai szimuláció')
self.w.setGeometry(100, 100, 1280, 768)
self.w.show()

# Background grids
gx = gl.GLGridItem()
gx.rotate(90, 0, 1, 0)
gx.translate(-10, 0, 0)
self.w.addItem(gx)

gy = gl.GLGridItem()
gy.rotate(90, 1, 0, 0)
gy.translate(0, -10, 0)
self.w.addItem(gy)

gz = gl.GLGridItem()
gz.translate(0, 0, -10)
self.w.addItem(gz)

self.calcMeanAnomaly()
self.calcEccentricAnomaly()
self.calcRealAnomaly()
self.calcPosition()

for obj in self.objects:
    if(obj == "SUN"):
        radius = self.objects[obj].d*100/AU
```

```
            else:
                radius = self.objects[obj].d*10000/AU

            sphere = gl.MeshData.sphere(rows=50, cols=50, radius=radius)

            self.traces[obj] = gl.GLMeshItem(
                meshdata=sphere,
                smooth=True,
                color=pg.glColor(self.objects[obj].color),
                glOptions="translucent",
                shader="shaded"
            )
            self.traces[obj].rotate(
                self.objects[obj].omega, 0, 0, 1, local=False)
            self.traces[obj].rotate(
                self.objects[obj].i, 0, 1, 0,
                local=False
            )
            self.w.addItem(self.traces[obj])
            text = pg.TextItem("asd")
        self.setPositions()

    # Középanomália meghatározása
    def calcMeanAnomaly(self):
        if(self.T_CURRENT is None):
            self.T_CURRENT = datetime.datetime.now()
        self.T_CURRENT += datetime.timedelta(days=1)

        for obj in self.objects:
            if obj == 'SUN':
                continue
```

```
        if(self.objects[obj].tau is None):
            self.objects[obj].tau = datetime.datetime.now()

        if (self.objects[obj].a is None or
                self.objects[obj].m is None):
            print("Value missing from object ", obj)
            print("Semi-major axis: ", self.objects[obj].a)
            print("Object's mass: ", self.objects[obj].m)
            return

        mu = G*(self.objects['SUN'].m+self.objects[obj].m)
        M = np.sqrt(
            mu/self.objects[obj].a**3)*(
                (self.T_CURRENT-self.objects[obj].tau).total_seconds()
        )
        self.objects[obj].M = M
    return

# Excentrikus anomália meghatározása
def calcEccentricAnomaly(self):
    for obj in self.objects:
        if obj == 'SUN':
            continue

        if (self.objects[obj].M is None or
                self.objects[obj].e is None):
            print("Values missing from object ", obj)
            print("Eccentricity: ", self.objects[obj].e)
            print("Mean Anomaly: ", self.objects[obj].M)
            return

        if self.objects[obj].E is None:
```

```
            self.objects[obj].E = 0.0

        def f(E, M=self.objects[obj].M, e=self.objects[obj].e):
            return M - E + e*np.sin(E)

        def df(E, e=self.objects[obj].e):
            return - 1 + e*np.cos(E)

        p_0 = self.objects[obj].M
        N = 10
        tol = 1e-5
        E = NewtonMethod(p_0, tol, N, f, df)

        self.objects[obj].E = E
    return

# Valódi anomália meghatározása
def calcRealAnomaly(self):
    for obj in self.objects:
        if obj == 'SUN':
            continue

        if (self.objects[obj].e is None or
                self.objects[obj].E is None):
            print("Values missing from object ", obj)
            print("Eccentricity: ", self.objects[obj].e)
            print("Eccentric Anomaly: ", self.objects[obj].E)
            return

        e = self.objects[obj].e
        E = self.objects[obj].E
        v = 2 * np.arctan(np.sqrt((1+e)/(1-e))*np.tan(E/2))
```

```
            self.objects[obj].v = v
        return

    # Pozíció meghatározása
    def calcPosition(self):
        for obj in self.objects:
            if(self.objects[obj].v is None):
                self.objects[obj].v = 0.0

            if(self.objects[obj].M is None):
                self.objects[obj].M = 0.0

            if(self.objects[obj].i is None):
                self.objects[obj].i = 0.0

            if(self.objects[obj].e is None):
                self.objects[obj].e = 0.0

            a = self.objects[obj].a
            e = self.objects[obj].e
            v = self.objects[obj].v

            r = (a*(1-e**2)) / (1+e*np.cos(v))

            x = r*np.cos(v)
            y = r*np.sin(v)

            self.objects[obj].r_o = self.objects[obj].r
            self.objects[obj].r = 10*np.array([x, y, 0])
        return
```

```
    def start(self):
        QtGui.QApplication.instance().exec_()

    def setPositions(self):
        for obj in self.objects:
            if self.objects[obj].r_o is None:
                self.objects[obj].r_o = np.array([0.0, 0.0, 0.0])
            delta_pos = self.objects[obj].r-self.objects[obj].r_o
            self.traces[obj].translate(*delta_pos/AU, local=True)
        return

    def update(self):
        self.calcMeanAnomaly()
        self.calcEccentricAnomaly()
        self.calcRealAnomaly()
        self.calcPosition()
        self.setPositions()

    def animation(self):
        timer = QtCore.QTimer()
        timer.timeout.connect(self.update)
        timer.start(20)
        self.start()

# m = mass [kg]
# d = diameter [m]
# a = semi major axis [m]
# e = eccentricity []
# i = inclination [deg]
# omega = argument of pericenter [deg]
# r_o = previous position [m]
# r = current position [m]
```

```
# v = real anomaly [rad]
# E = eccentric anomaly [rad]
# M = mean anomaly [rad]
DATA = {
    "SUN": {
        "m": 1.9891e30,
        "d": 1.39268e9,
        "a": 0,
        "e": None,
        "i": 0.0,
        "tau": None,
        "omega": 0.0,
        "r": None,
        "r_o": None,
        "v": None,
        "E": None,
        "M": None,
        "color": np.array([255, 255, 0])
    },
    "MERCURY": {
        "m": 0.3302e24,
        "d": 4.879e6,
        "a": 57.9e9,
        "e": 0.205,
        "i": 7.0,
        "tau": None,
        "omega": 29.124,
        "r": None,
        "r_o": None,
        "v": None,
        "E": None,
        "M": None,
```

```
    "color": np.array([148, 148, 148])
},
"VENUS": {
    "m": 4.8685e24,
    "d": 12.104e6,
    "a": 108.2e9,
    "e": 0.007,
    "i": 3.4,
    "tau": None,
    "omega": 54.884,
    "r": None,
    "r_o": None,
    "v": None,
    "E": None,
    "M": None,
    "color": np.array([255, 176, 31])
},
"EARTH": {
    "m": 5.9736e24,
    "d": 12.756e6,
    "a": 149.6e9,
    "e": 0.017,
    "i": 0.0,
    "tau": None,
    "omega": 114.20783,
    "r": None,
    "r_o": None,
    "v": None,
    "E": None,
    "M": None,
    "color": np.array([90, 162, 250])
},
```

```
"MARS": {
    "m": 0.64171e24,
    "d": 6.792e6,
    "a": 227.9e9,
    "e": 0.094,
    "i": 1.9,
    "tau": None,
    "omega": 286.502,
    "r": None,
    "r_o": None,
    "v": None,
    "E": None,
    "M": None,
    "color": np.array([255, 0, 0])
},
"JUPITER": {
    "m": 1898e24,
    "d": 142.984e6,
    "a": 778.6e9,
    "e": 0.049,
    "i": 1.3,
    "tau": None,
    "omega": 273.867,
    "r": None,
    "r_o": None,
    "v": None,
    "E": None,
    "M": None,
    "color": np.array([138, 67, 0])
},
"SATURN": {
    "m": 568e24,
```

```
        "d": 120.536e6,
        "a": 1433.5e9,
        "e": 0.057,
        "i": 2.5,
        "tau": None,
        "omega": 339.392,
        "r": None,
        "r_o": None,
        "v": None,
        "E": None,
        "M": None,
        "color": np.array([252, 205, 119])
    },
    "URANUS": {
        "m": 86.8e24,
        "d": 51.118e6,
        "a": 2872.5e9,
        "e": 0.046,
        "i": 0.8,
        "tau": None,
        "omega": 96.998857,
        "r": None,
        "r_o": None,
        "v": None,
        "E": None,
        "M": None,
        "color": np.array([101, 141, 150])
    },
    "NEPTUNE": {
        "m": 102e24,
        "d": 49.528e6,
        "a": 4495.1e9,
```

```
        "e": 0.011,
        "i": 1.8,
        "tau": None,
        "omega": 276.336,
        "r": None,
        "r_o": None,
        "v": None,
        "E": None,
        "M": None,
        "color": np.array([77, 145, 240])
    },
    "PLUTO": {
        "m": 0.0146e24,
        "d": 2.370e6,
        "a": 5906.4e9,
        "e": 0.244,
        "i": 17.2,
        "tau": None,
        "omega": 113.834,
        "r": None,
        "r_o": None,
        "v": None,
        "E": None,
        "M": None,
        "color": np.array([179, 139, 55])
    },
}

if __name__ == '__main__':
    TwoBodyOrbits(DATA).animation()
```

NYILATKOZAT

Alulírott ..... Fizika BSc szakos hallgató (ETR azonosító: .....) a .....

.....

című szakdolgozat szerzője fegyelmi felelősségem tudatában kijelentem, hogy dolgozatom önálló munkám eredménye, saját szellemi termékem, abban a hivatkozások és idézések általános szabályait következetesen alkalmaztam, mások által írt részeket a megfelelő idézés nélkül nem használtam fel.

Szeged ,..... év ..... hó ..... nap

.....

a hallgató aláírása