ÖNSZERV HF2

```
%pylab inline
from scipy.optimize import curve_fit # Az illesztéshez használt
függvény
from numpy.fft import *             # Fourier-analízishez használt
rutinok
from scipy.signal import spectrogram # Spektrogramm készítő függvény
from scipy.interpolate import interp1d # Interpoláció
import scipy.integrate as integrate # Integrálás
import sys
from scipy.integrate import odeint
import matplotlib.pyplot as plt
from matplotlib.patches import Circle
import imageio
```

```
Populating the interactive namespace from numpy and matplotlib
```

1. feladat

Írjuk fel a kettős inga az ideális (surlódás nélküli) egyenleteit!

Megoldás

Az Euler-Lagrange egyenletek:

$$(m_1+m_2)\cdot l_1\cdot\theta_1''+m_2\cdot l_2\cdot\theta_2''\cdot cos(\theta_2-\theta_1)=m_2\cdot l_2\cdot\theta_2'^2\cdot sin(\theta_2-\theta_1)-(m_1+m_2)\cdot g\cdot sin(\theta_1)$$

$$l_2\cdot\theta_2''+l_1\cdot\theta_1''\cdot cos(\theta_2-\theta_1)=-l_1\cdot\theta_1'^2\cdot sin(\theta_2-\theta_1)-g\cdot sin(\theta_2)$$

Ezekből célszerű kifejezni úgy θ_1''-t és θ_2''-t, hogy ne függjenek egymástól expliciten. A második egyenletből gyártsunk $\theta_2''(\theta_1'')$ összefüggést, majd helyettesítsük az elsőbe.

$$\theta_2''=\frac{1}{l_2}\cdot(-l_1\cdot\theta_1'^2\cdot sin(\theta_2-\theta_1)-g\cdot sin(\theta_2)-l_1\cdot\theta_1''\cdot cos(\theta_2-\theta_1))$$

$$\theta_1''\cdot((m_1+m_2)\cdot l_1+m_2\cdot l_2\cdot cos(\theta_2-\theta_1)\cdot\frac{-l_1\cdot cos(\theta_2-\theta_1)}{l_2})=m_2\cdot l_2\cdot\theta_2'^2\cdot sin(\theta_2-\theta_1)-(m_1+m_2)\cdot g\cdot sin(\theta_1)-m_2\cdot l_2$$

$$\theta_1''=\frac{m_2\cdot sin(\theta_2-\theta_1)\cdot[l_2\cdot\theta_2'^2+l_1\cdot\theta_1'^2\cdot cos(\theta_2-\theta_1)]+m_2\cdot cos(\theta_2-\theta_1)\cdot g\cdot sin(\theta_2)-(m_1+m_2)\cdot g\cdot sin(\theta_1)}{l_1\cdot(m_1+m_2\cdot sin^2(\theta_2-\theta_1))}$$

Hasonlóképp megcsinálható ez θ_2''-re is.

$$\theta_2''=(m_1+m_2)\cdot(l_1\theta_1'^2\cdot sin(\theta_2-\theta_1)-g\cdot sin(\theta_2)+g\cdot sin(\theta_1)\cdot cos(\theta_2-\theta_1))+m_2\cdot l_2\cdot\theta_2'^2\cdot sin(\theta_2-\theta_1)\cdot cos(\theta_2-\theta_1)¿$$

Ezeket az egyenleteket már tudjuk kezelni numerikusan.

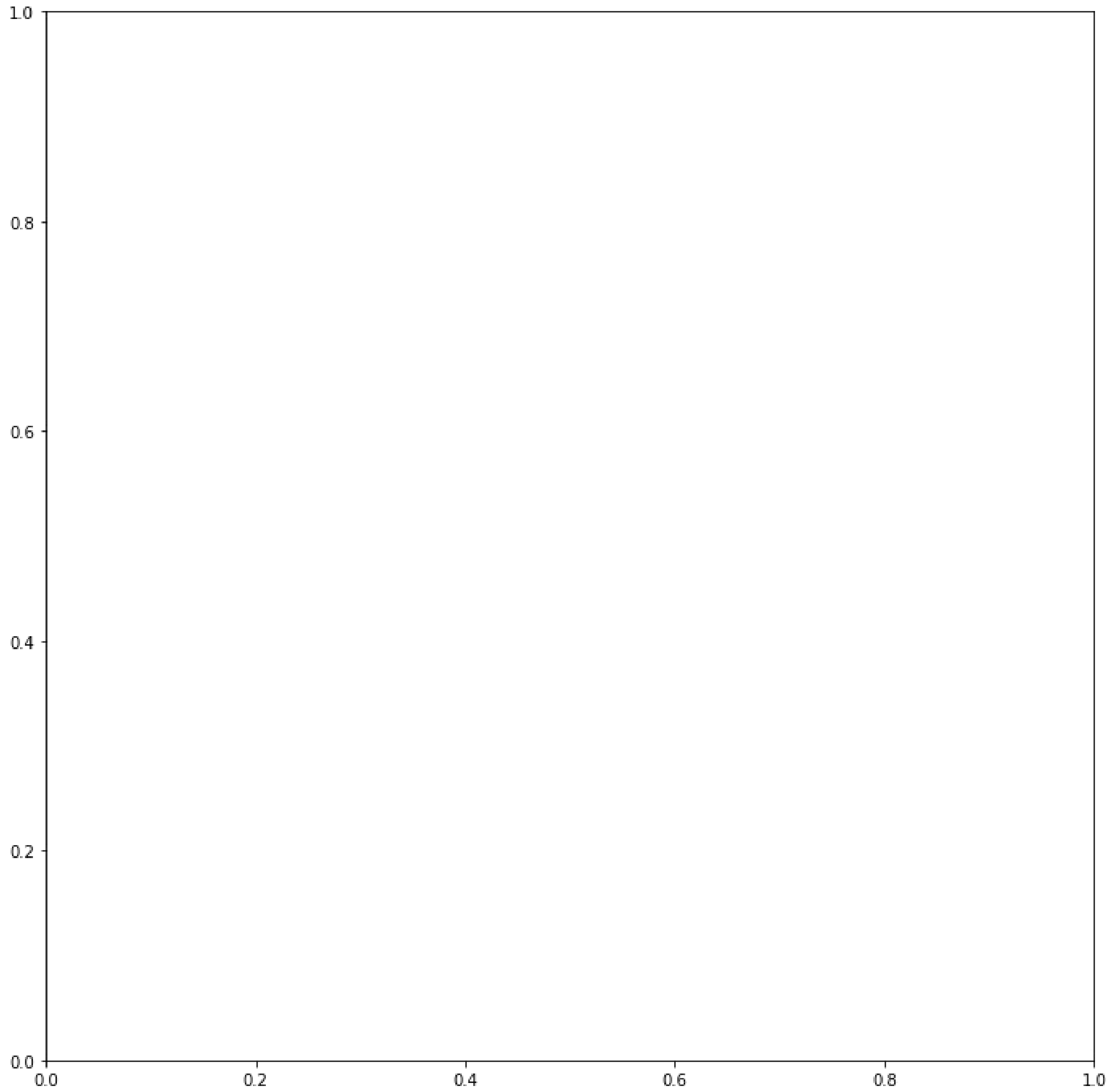
2. feladat

Írjuk meg tetszőleges programnyelvben az ideális kettős inga szimulációját!

Megoldás

```
g = 9.81 # m/s^2
# Az ingák hosszai :
l1 = 1 # m
l2 = 2 # m
#Az ingák tömegei:
m1 = 1 # kg
m2 = 1 # kg

def dif(y, t, l1, l2, m1, m2): # visszaadja y deriváltjait
    # y alakja : th1, th1', th2, th2'
    # return : y' = a, b, c, d
    th1, dth1, th2, dth2 = y
    a = dth1
    c = dth2

    b = (m2*g*np.sin(th2)*np.cos(th1-th2) - m2*sin(th1-
th2)*(l1*a**2*np.cos(th1-th2) + l2*c**2) - (m1+m2)*g*np.sin(th1)) /
(l1 * (m1 + m2*(np.sin(th1-th2))**2))
    d = ((m1+m2)*(l1*a**2*np.sin(th1-th2) - g*np.sin(th2) +
g*np.sin(th1)*np.cos(th1-th2)) + m2*l2*c**2*np.sin(th1-
th2)*np.cos(th1-th2)) / (l2 * (m1 + m2*(np.sin(th1-th2))**2))
    return a, b, c, d

tmax, dt = 25, 0.01 # meddig és milyen lépésközzel
t = np.arange(0, tmax+dt, dt) # mint a tömb
y0 = np.array([3*np.pi/7, 0, 3*np.pi/4, 0]) # kezdőfeltételek a dif
egyenletekhez
        # y alakja : th1, th1', th2, th2'

y = odeint(dif, y0, t, args=(l1, l2, m1, m2)) # az integrálás

th1, th2 = y[:,0], y[:,2]

# Déscartes-koordináták
x1 = l1 * np.sin(th1)
y1 = -l1 * np.cos(th1)
x2 = x1 + l2 * np.sin(th2)
y2 = y1 - l2 * np.cos(th2)

R = 0.1 # a körök átmérői

def kep(i):
    ax.plot([0, x1[i], x2[i]], [0, y1[i], y2[i]], lw=2, c='k') # a két
rúd
    c0 = Circle((0, 0), R/2, fc='k', zorder=10) # a felsfüggesztés
    c1 = Circle((x1[i], y1[i]), R, fc='b', ec='b', zorder=10) # az
```

```
első test
    c2 = Circle((x2[i], y2[i]), R, fc='r', ec='r', zorder=10) # a
második test
    ax.add_patch(c0)
    ax.add_patch(c1)
    ax.add_patch(c2)


    # a felfüggesztésre centráljuk a képet
    ax.set_xlim(-l1-l2-R, l1+l2+R)
    ax.set_ylim(-l1-l2-R, l1+l2+R)
    ax.set_aspect('equal', adjustable='box')
    plt.axis('off')
    plt.savefig("img{:d}.png".format(i//di), dpi=72)
    plt.cla()

fps = 8
di = int(1/fps/dt)
fig = plt.figure(figsize=(16, 12), dpi=72)
ax = fig.add_subplot(111)

for i in range(0, t.size, di):
    kep(i)

images = []
for i in range(t.size//di):
    s = "img"+str(i)+".png"
    images.append(imageio.imread(s))
imageio.mimsave("movie1112.gif", images)
```




```
t.size
```

```
2501
```

3. feladat

A két szögre és a két szögsebességre összesen 4 egyenletet kapunk. Az energia megmaradása miatt elegendő ezek közül csak három viselkedését követni. A mozgás követéséhez a 3 dimenziós pálya helyett elegendő annak csak egyik metszetét (ún. Poincaré metszet) követni. A legegyszerűbb metszet, amikor az alsó inga szögét és szögsebességét ábrázoljuk egymás függvényében azokban a pillanatokban, amikor a felső inga pozitív irányban lengve éppen áthalad a függőlegesen (azaz a 0 szögkitérésen).

Készítsünk ilyen ábrákat különböző paraméter-beállítások mellett. Próbáljunk kaotikus és periodikus pályákat is találni a fázistérben.

A megoldást dokumentáljuk néhány saját ábrával és magyarázattal!

Megoldás

Az előzőekhez képest szignifikánsan tovább kell futtatni a numerikus számolást, hogy elég pont legyen a Poincaré-metszeten. Az előző megoldás önmagában szemléltetés gyanánt jó volt, de 2 nagy technikai problémával is meg kell küzdeni ebben a feladatrészben. Az egyik, hogy a .gif-be írásnak sok nagyságrenddel több időre van szüksége. A másik probléma, hogy az energiamegmaradás nem teljesül egzaktul, ha numerikusan számolunk.

Az első probléma miatt nem célszerű .gif-be elmenteni a megoldást. A második probléma miatt szigorúbbá kell tenni az integrálást - kisebb lépésközzel kell integrálni. Az integrálás futási ideje is meghízik most, mivel sokkal tovább integrálunk, ráadásul kisebb lépésközzel.

```
tmax, dt = 6000, 0.001 # meddig és milyen lépésközzel
t = np.arange(0, tmax+dt, dt) # mint a tömb
y0 = np.array([3*np.pi/7, 0, 3*np.pi/4, 0]) # kezdőfeltételek a dif
egyenletekhez
        # y alakja : th1, th1', th2, th2'

y = odeint(dif, y0, t, args=(l1, l2, m1, m2)) # az integrálás


th1_p, th2_p = y[:,0], y[:,2]
dth1_p, dth2_p = y[:,1], y[:,3]
```

```
len(th1_p), th1_p
```

```
(6000001,
 array([  1.34639685,   1.34639236,   1.34637887, ..., -58.7080959 ,
        -58.70763323, -58.70715583]))
```

Ne a függőlegesen való áthaladás pontatlanság toleranciáját figyljük, hanem azt, amikor előjelet vált θ_1. A toleranciát nehéz dinamikusan állítani - áthaladásokat kihagyhatunk, illetve egy áthaladásról több pont is false-positive lesz, így mindkét irányban rontunk a (jó pontok / rossz pontok) arányon. Ráadásul amikor egy áthaladásnál van false-positive, akkor általában sok van - ez drasztikusan lassítja a plotolást.

```
u = [] # ide mennek a szögek
v = [] # ide mennek a szögsebességek


for i in range(len(th1_p)-1):
    if ((th1_p[i]<0) and (th1_p[i+1]>0)):
        u.append(th2_p[i])
        v.append(dth2_p[i])
        print(i)
    if ((th1_p[i+1]<0) and (th1_p[i]>0)):
        u.append(th2_p[i])
        v.append(dth2_p[i])
        print(i)
```

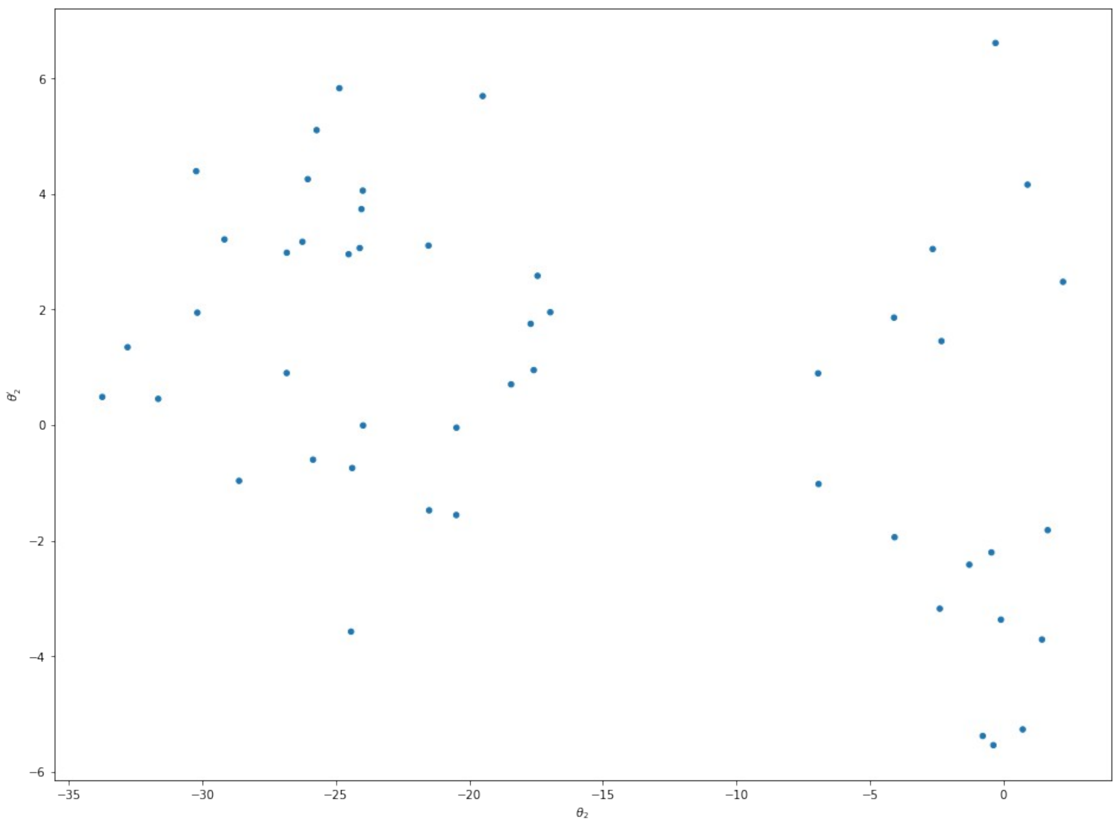
668
833
1261
2803
4120
4672
5090
6127
7024
8778
9677
10699
11305
11905
13457
14649
15551
17236
78545
79511
91725
92961
93679
94712
95257
95842
97076
98113
99712
101257
101547
101723
103268
105146
105217
105554
107129
108728
109846
110918
112301
113915
114509
114799
115877
116649
525680
526870
527768

Ez azért érdekes, mert a 116649-es index után átfordul az inga, tehát θ_1 már nem a $[-\pi, \pi]$ intevallumon fog mozogni, és az 525680-es indexig pedig pontosan ugyanannyiszor fordult vissza, mint amennyit átfordult - mindezt úgy, hogy az átfordulások és a visszafordulások különbsége nem érintette a 0-t. Ez valószínűtlennek tűnik, de a magyarázat méginkább az. A "movie_alt.gif"-be kiírattam az előbbi index utáni pár másodpercet, az inga több, mint ötször (!) átfordul egymás után.

```
len(u),len(v), len(th1_p)
```

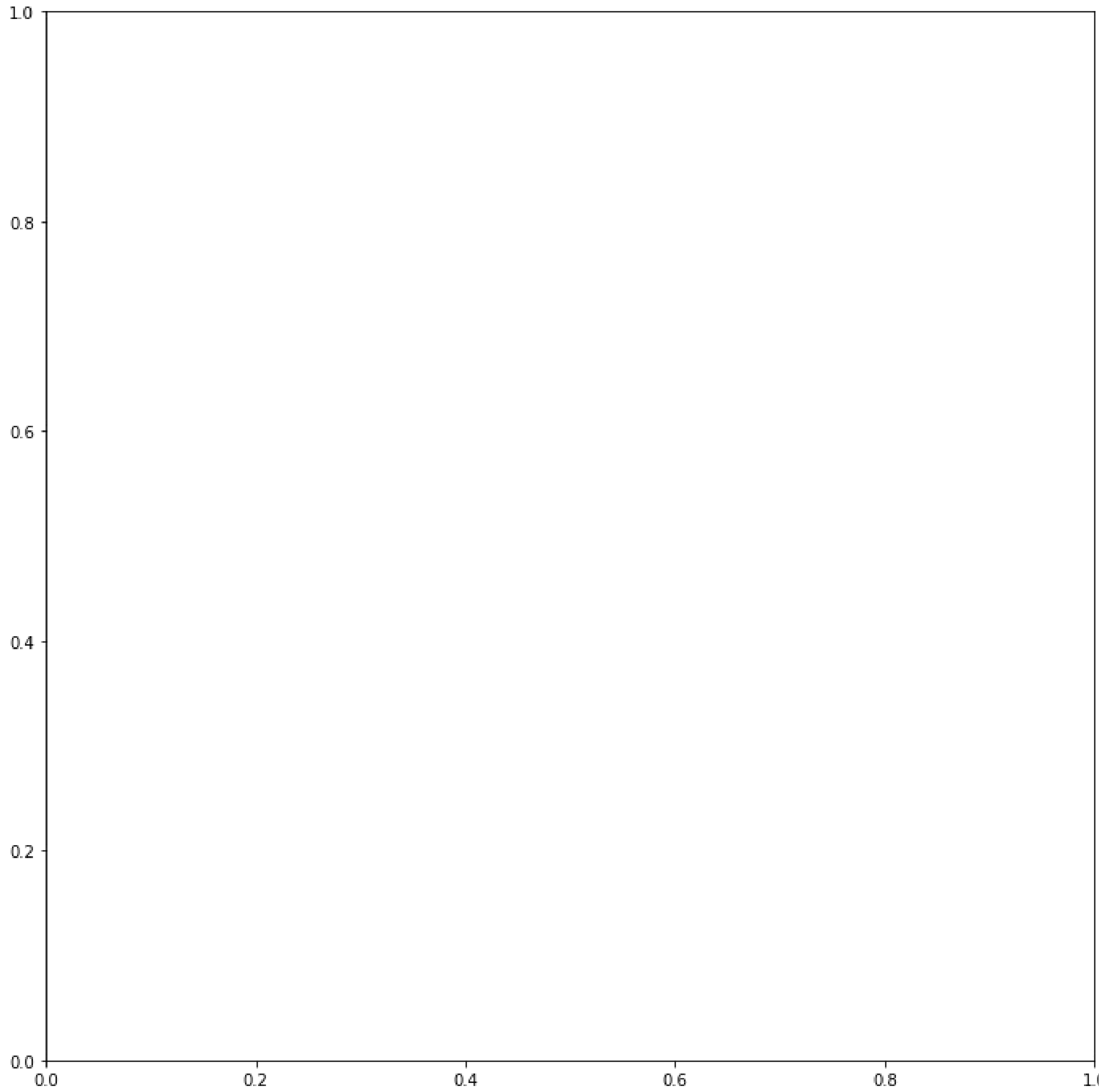
```
(49, 49, 6000001)
```

```
figsize(16,12)
plt.scatter(u, v, np.ones(len(u))*20, marker="o")
plt.xlabel("$ \\theta_2 $")
plt.ylabel("$ \\theta_2' $")
plt.show()
```




```
x1_p = l1 * np.sin(th1_p)
y1_p = -l1 * np.cos(th1_p)
x2_p = x1_p + l2 * np.sin(th2_p)
y2_p = y1_p - l2 * np.cos(th2_p)

def kep_p(i):
    ax.plot([0, x1_p[i], x2_p[i]], [0, y1_p[i], y2_p[i]], lw=2, c='k')
# a két rúd
    c0 = Circle((0, 0), R/2, fc='k', zorder=10) # a felsfüggesztés
```

```
    c1 = Circle((x1_p[i], y1_p[i]), R, fc='b', ec='b', zorder=10) # az első test
    c2 = Circle((x2_p[i], y2_p[i]), R, fc='r', ec='r', zorder=10) # a második test
    ax.add_patch(c0)
    ax.add_patch(c1)
    ax.add_patch(c2)


    # a felfüggesztésre centráljuk a képet
    ax.set_xlim(-l1-l2-R, l1+l2+R)
    ax.set_ylim(-l1-l2-R, l1+l2+R)
    ax.set_aspect('equal', adjustable='box')
    plt.axis('off')
    plt.savefig("img_p{:d}.png".format(i//di), dpi=72)
    plt.cla()

fps = 5
di = int(1/fps/dt)
fig = plt.figure(figsize=(16, 12), dpi=72)
ax = fig.add_subplot(111)

innen = 116649
ennyit = 250*di

for i in range(innen, innen+ennyit, di):
    kep_p(i)

images = []
for i in range(innen//di, (innen+ennyit)//di):
    s = "img_p"+str(i)+".png"
    images.append(imageio.imread(s))
imageio.mimsave("movie1112_alt.gif", images)
```



A kicsit fentebb tárgyalt okok miatt korrigálni kell a képletet, hogy érzékeny legyen $-\pi$ és π többszöröseire is. Az akarjunk, hogy a π többszörösei 0-t adjanak vissza, a $[-\pi, -2\pi]$ intervallumban lévő számok a $[\pi, 0]$ intervallumba képződjenek megfelelően, és hasonlóan a pozitív oldalon.

Ez általánosítható minden hasonló intervallumra :

- ha negatív irányban $[-\pi, -2\pi]$, $[-3\pi, -4\pi]$, $[-5\pi, -6\pi]$, ... intervallumban van egy szám, akkor a $[\pi, 0]$ intervallumba képződjön
- ha negatív irányban $[0, -\pi]$, $[-2\pi, -3\pi]$, $[-4\pi, -5\pi]$, ... intervallumban van egy szám, akkor a $[-\pi, 0]$ intervallumba képződjön
- ha pozitív irányban $[\pi, 2\pi]$, $[3\pi, 4\pi]$, $[5\pi, 6\pi]$, ... intervallumban van egy szám, akkor a $[-\pi, 0]$ intervallumba képződjön
- ha pozitív irányban $[0, \pi]$, $[2\pi, 3\pi]$, $[4\pi, 5\pi]$, ... intervallumban van egy szám, akkor a $[\pi, 0]$ intervallumba képződjön Ennek a kompakt matematika leírása egy x szögre:

$$x = (x \% \pi) - \pi \cdot ((x / ¿ \pi) \% 2)$$

Ez a $[0, \pi]$ és $[0, -\pi]$ intervallumokat is megfelelően saját magukba képzi. Ráadásul nem rontja el, hogy csak az előjelváltást kell figyelni.

```
u = [] # ide mennek a szögek
v = [] # ide mennek a szögsebességek

for i in range(len(th1_p)-1):
    buff1 = (th1_p[i] % pi)-pi*((th1_p[i]//pi)%2)
    buff2 = (th1_p[i+1] % pi)-pi*((th1_p[i+1]//pi)%2)
    if ((buff1<0) and (buff2>0)):
        u.append(th2_p[i])
        v.append(dth2_p[i])
#         print(i)
    if ((buff2<0) and (buff1>0)):
        u.append(th2_p[i])
        v.append(dth2_p[i])
#         print(i)
```

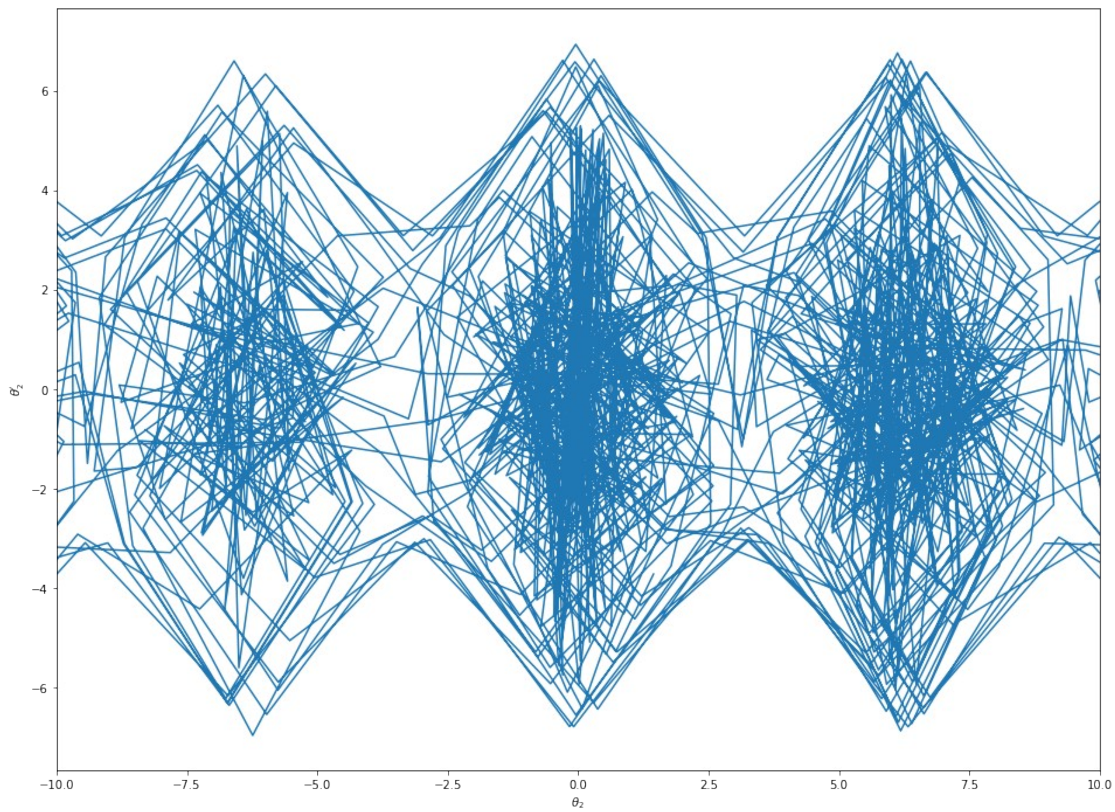
```
len(u)
```

```
6251
```

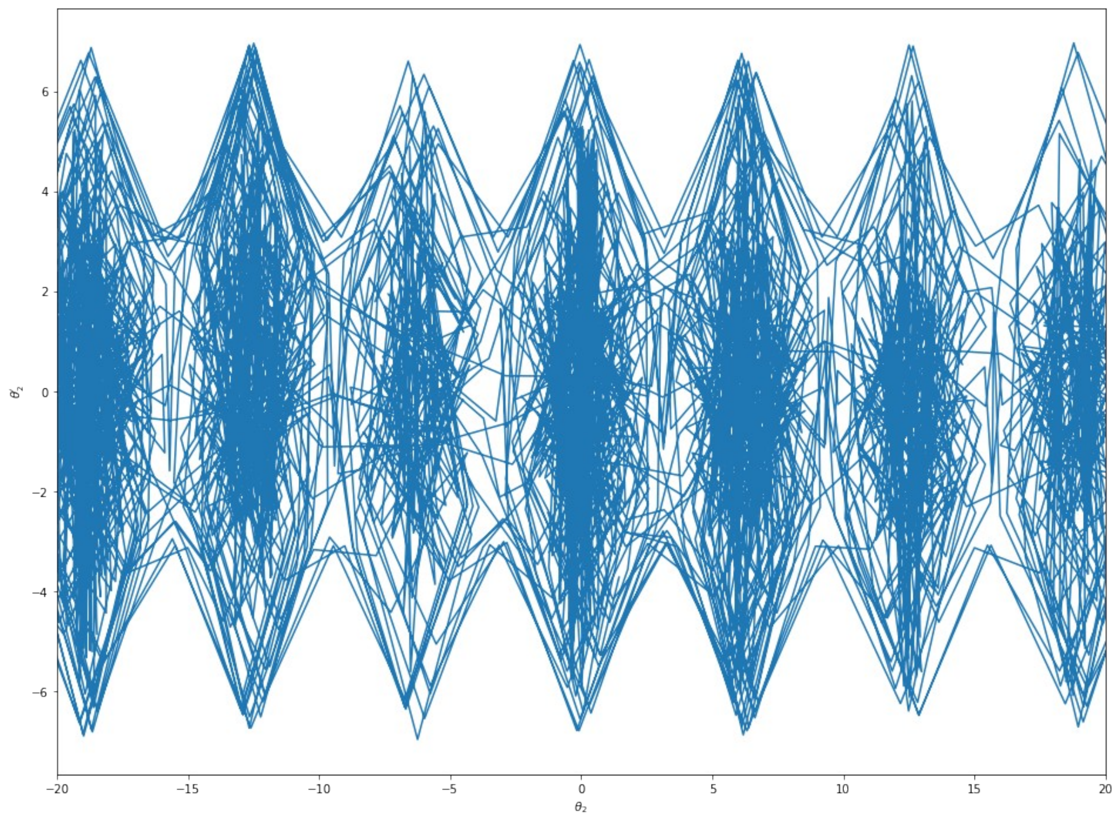
Az már sokkal hihetőbb eredmény, hogy átlagosan kb másodpercenként egyszer érinti a belső inga a függőlegest. Nézzük a jó Poincaré-metszetet.

```
figsize(16,12)
plt.scatter(u, v, np.ones(len(u))*20, marker="o")
plt.xlabel("$ \\theta_2 $")
plt.ylabel("$ \\theta_2' $")
plt.show()
```

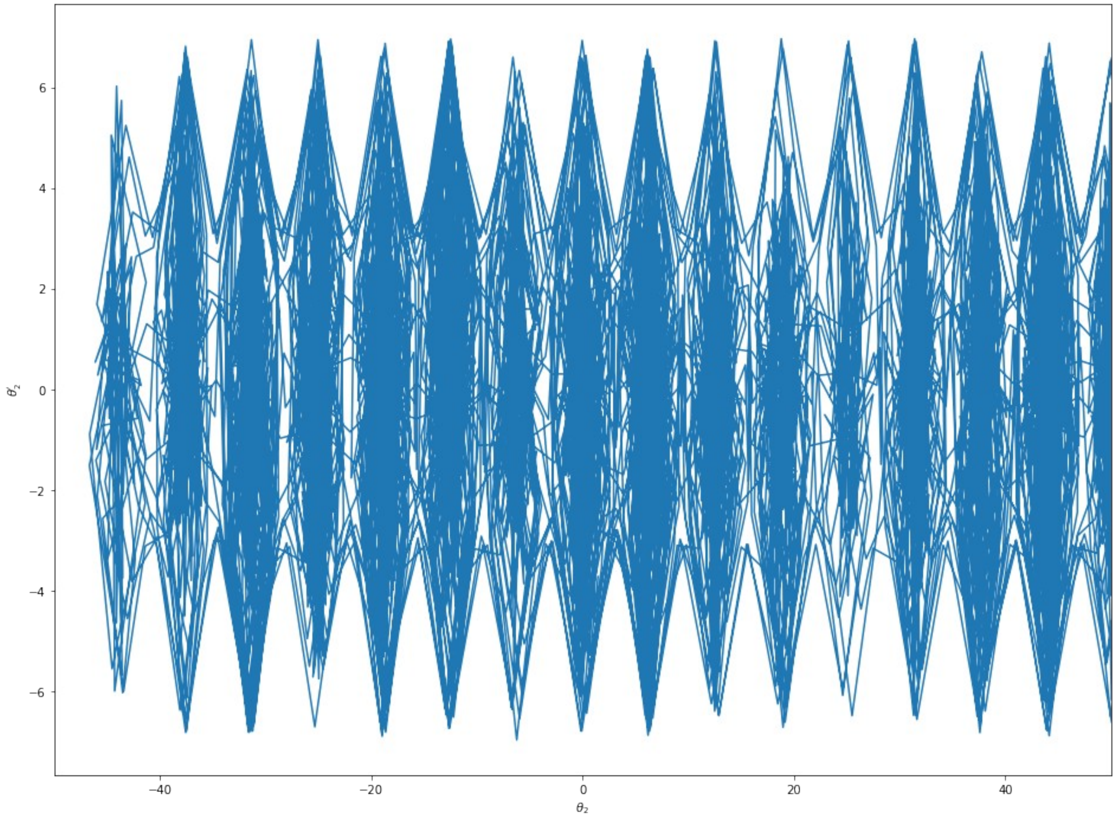

```
figsize(16,12)
plt.plot(u, v)
plt.xlim(-10,10)
plt.xlabel("$ \\theta_2 $")
plt.ylabel("$ \\theta_2' $")
plt.show()
```



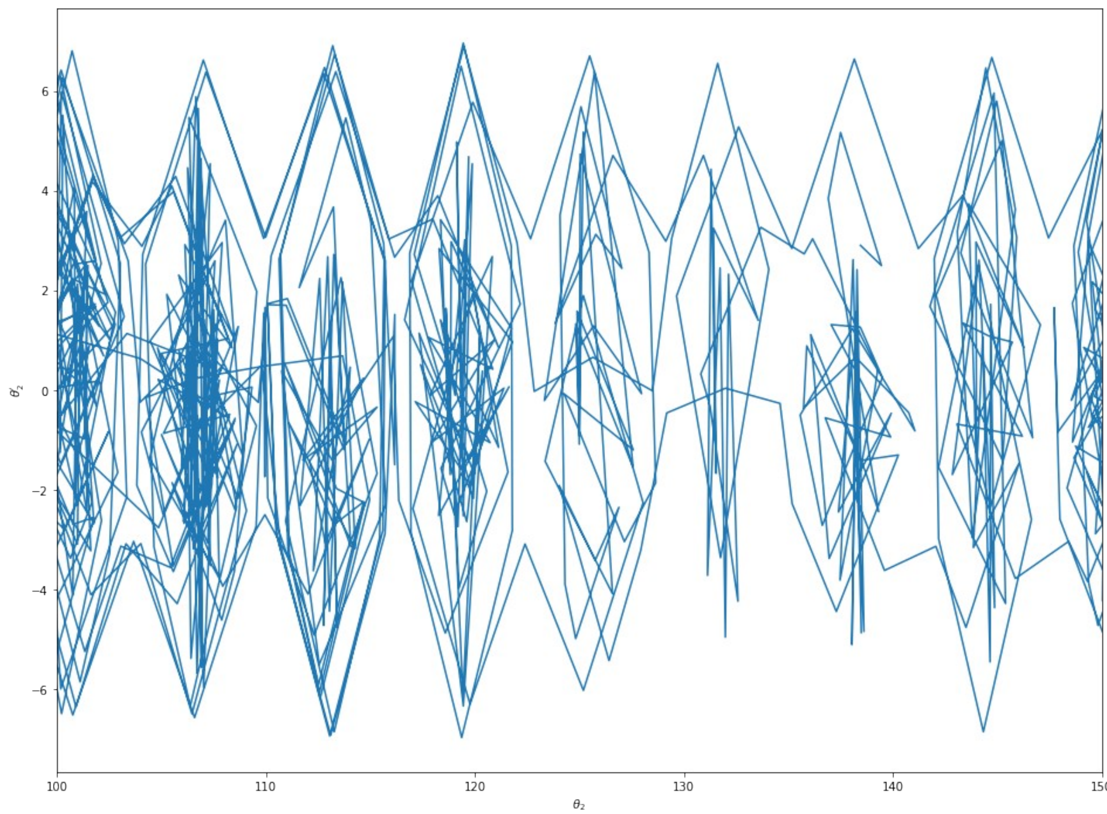

```
figsize(16,12)
plt.plot(u, v)
plt.xlim(-20,20)
plt.xlabel("$ \\theta_2 $")
plt.ylabel("$ \\theta_2' $")
plt.show()
```



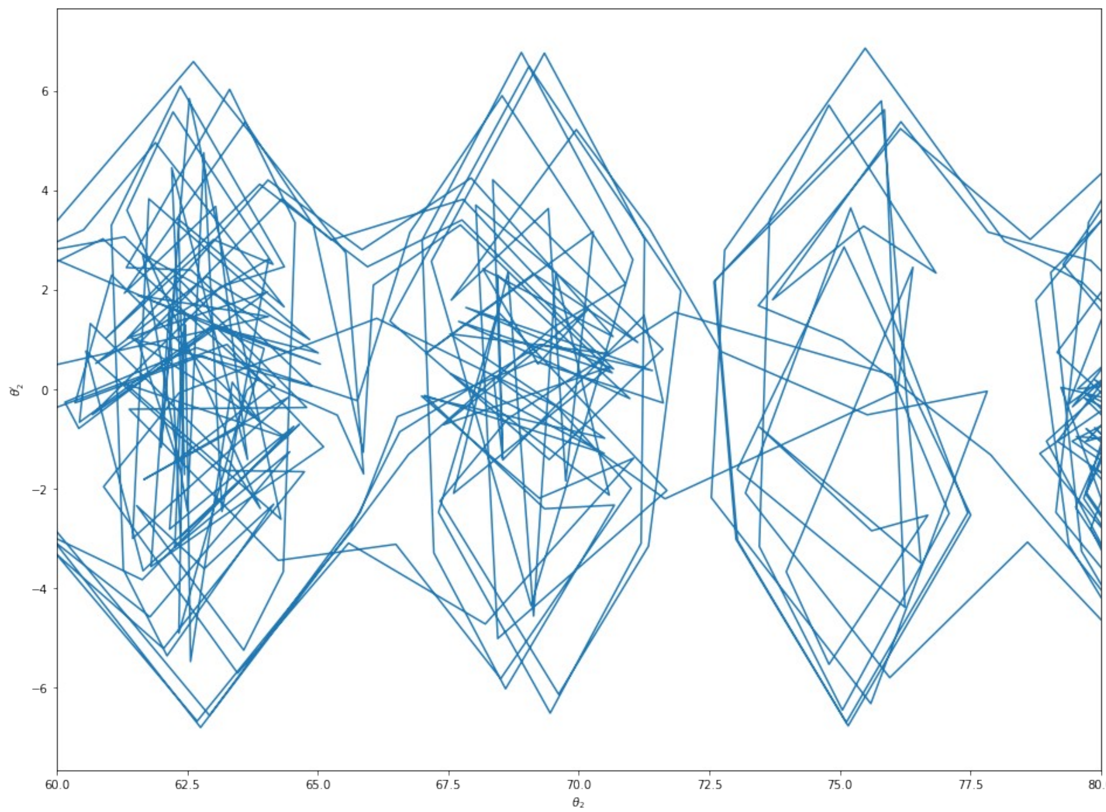

```
figsize(16,12)
plt.plot(u, v)
plt.xlim(-50,50)
plt.xlabel("$ \\theta_2 $")
plt.ylabel("$ \\theta_2' $")
plt.show()
```




```
figsize(16,12)
plt.plot(u, v)
plt.xlim(100,150)
plt.xlabel("$ \\theta_2 $")
plt.ylabel("$ \\theta_2' $")
plt.show()
```




```
figsize(16,12)
plt.plot(u, v)
plt.xlim(60,80)
plt.xlabel("$ \\theta_2 $")
plt.ylabel("$ \\theta_2' $")
plt.show()
```



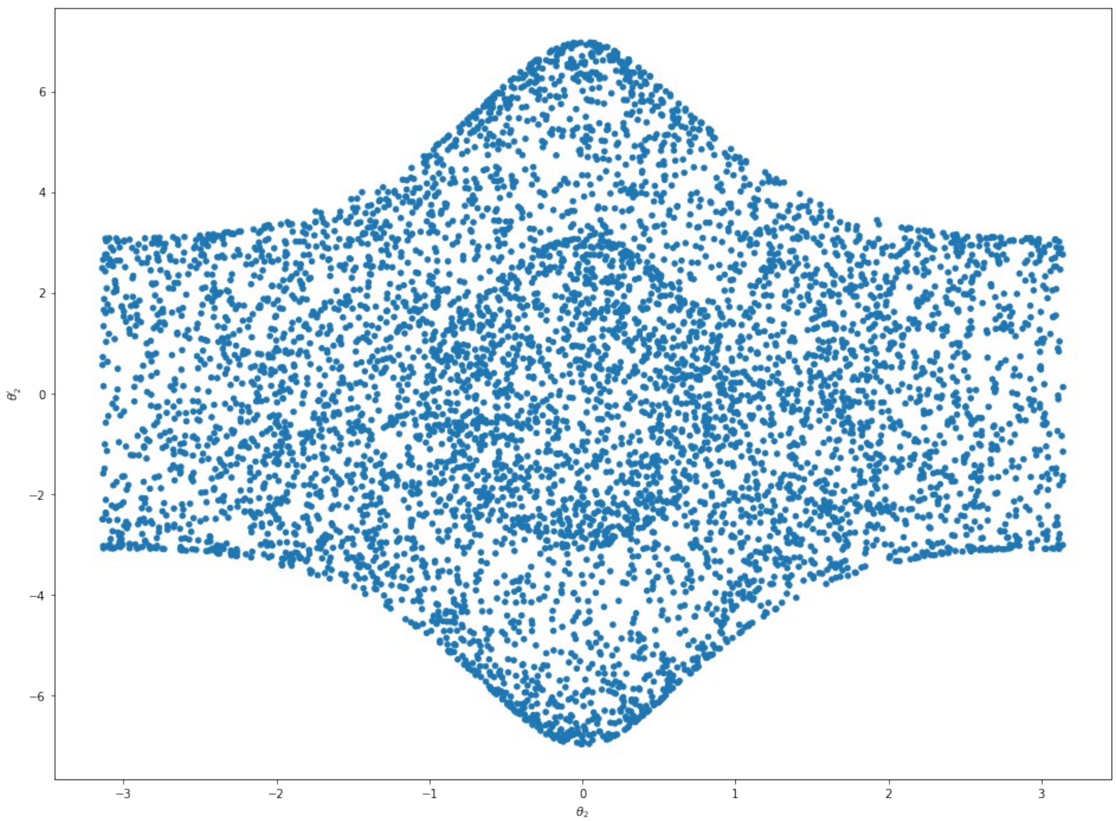
Fentről lefelé haladva egyre kaotikusabb mozgásokat láthatunk (részben azért, mert a nagyobb szögek előfordulása ritkább, mert ehhez sokszor át kellett fordulnia az ingának - ezek után azt is várjuk, hogy valamennyivel kaotikusabban mozogjon). Felfedezhetünk zárt görbéket ("zárt") a fázistérben. Ez θ_1-enkénti metszet, az energiamegmaradás miatt teljesülnek rá fázistérre jelemző tulajdonságok, mint a zárt görbe periódus jellege. Egy ilyen zárt görbe csak a második inga mozgására nézve tekinthető periódusnak. A modell miatt egy zárt görbén végighaladva nem feltétlen (sőt, általában nem - ahogy a cikázás mutatja) egymás után következnek a pontok. A második inga "próbál" periódikus maradni, de a "periódusosságának mértéke" még így is kaotikus, és a rendszer olyan erősen kezdőfeltétel függő, hogy nagyon könnyen átlendülhet egy másik "metastabil" gombócba. Megjegyzendő, hogy a sorban összekötött pontok diagramján az egyik gombócból a másikba való átlendülés sokszor egy hullámos burkoló görbe jellegű vonalat ad a mozgásnak - sorban vannak vonzási csomópontok a fázistérben, és hajlamosabb ezektől függőlegesen távolabb átlendülni.

Ezt azért látjuk, mert általában azt várjuk, hogy nagyobb szögsebesség kell neki, hogy átlendüljön, és ha átlendül, nehezebben lendül ki egy $[n\cdot\pi,(n+1)\cdot\pi]$ tartományból. Nézzük mi történik, ha a releváns szöget nézzük - azaz ugyanúgy szorítsuk be θ_2-t is $[-\pi,\pi]$ tartományba.

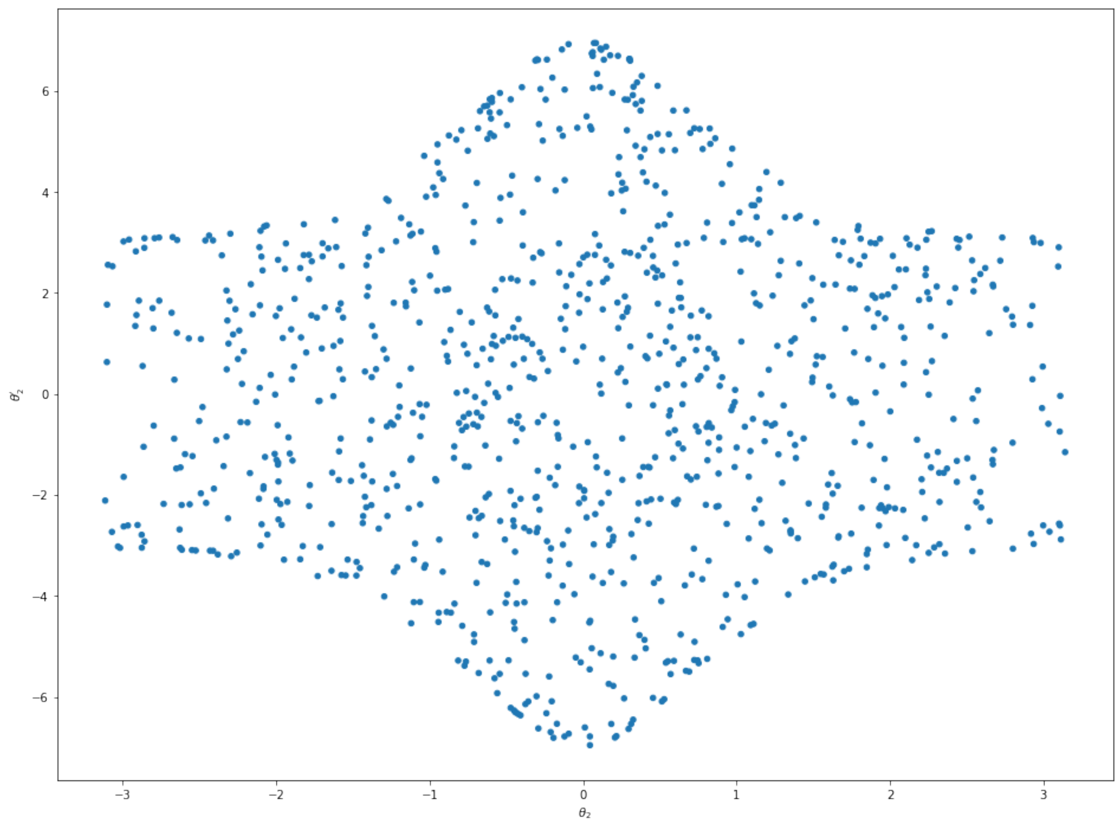
Megjegyzendő, hogy oda-vissza is tud lendülni a második inga, továbbá azért ekkora a cikázás, mert diszkréten mintavételezünk, nem pedig track-elünk. Ha például egy részecskegyorsítóban kijelölünk egy nyalábra transzverz síkot, és ugyanilyen metszetet készítünk egy lokális (x, x') fázistér-koordinátáról, a síkon való áthaladáskor gyűjtött pontok lehet egy ellipszisen lesznek (oszcillál x irányban), de várhatóan nem sorban egymás után.

```
u0 = []
for i in range(len(u)):
    u0.append((u[i] % pi)-pi*((u[i]//pi)%2))
```

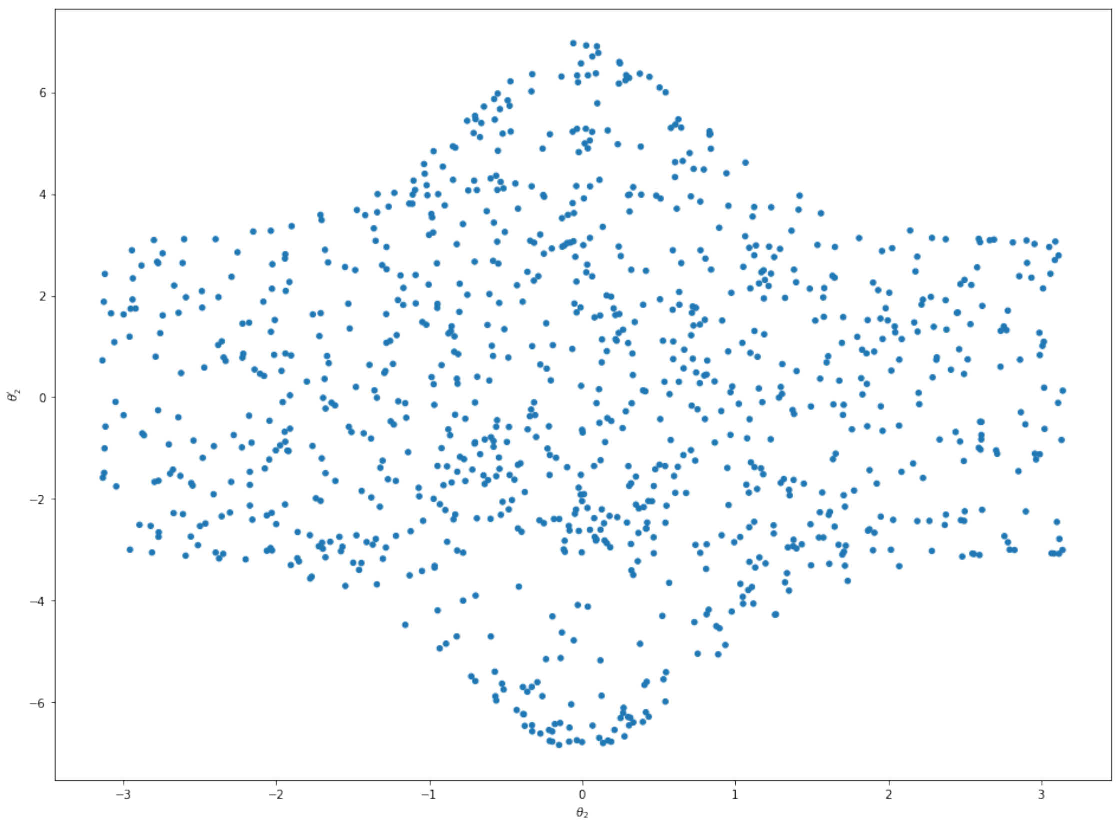
```
figsize(16,12)
plt.scatter(u0, v, np.ones(len(u))*20, marker="o")
plt.xlabel("$ \\theta_2 $")
plt.ylabel("$ \\theta_2' $")
plt.show()
```



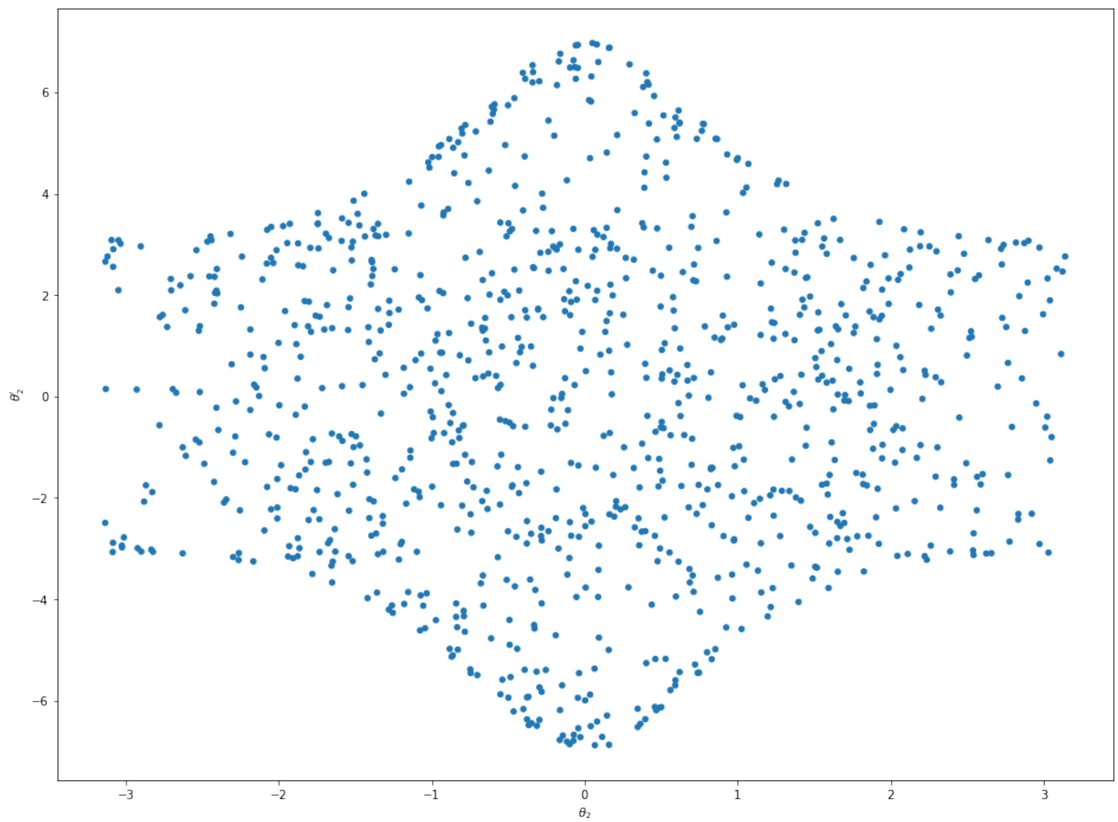

```
figsize(16,12)
plt.scatter(u0[0:1000], v[0:1000], np.ones(len(u[0:1000]))*20,
marker="o")
plt.xlabel("$ \\theta_2 $")
plt.ylabel("$ \\theta_2' $")
plt.show()
```



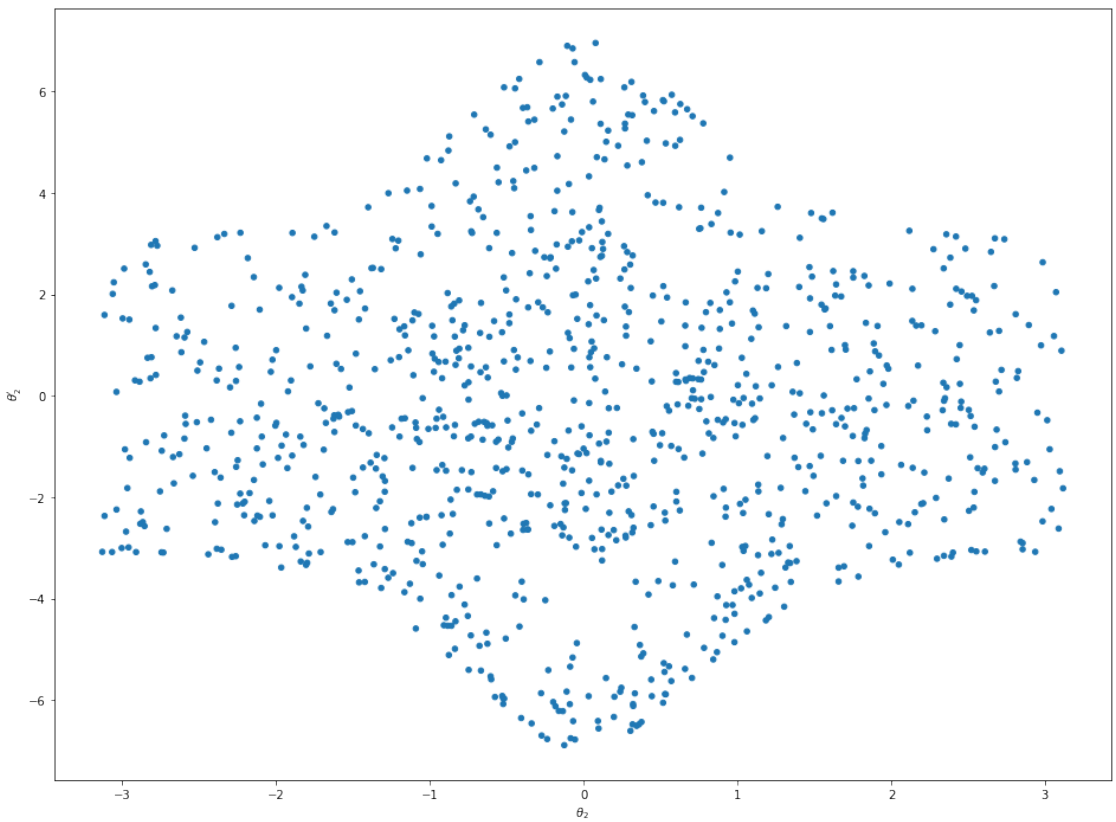

```
figsize(16,12)
plt.scatter(u0[1000:2000], v[1000:2000],
np.ones(len(u[1000:2000]))*20, marker="o")
plt.xlabel("$ \\theta_2 $")
plt.ylabel("$ \\theta_2' $")
plt.show()
```



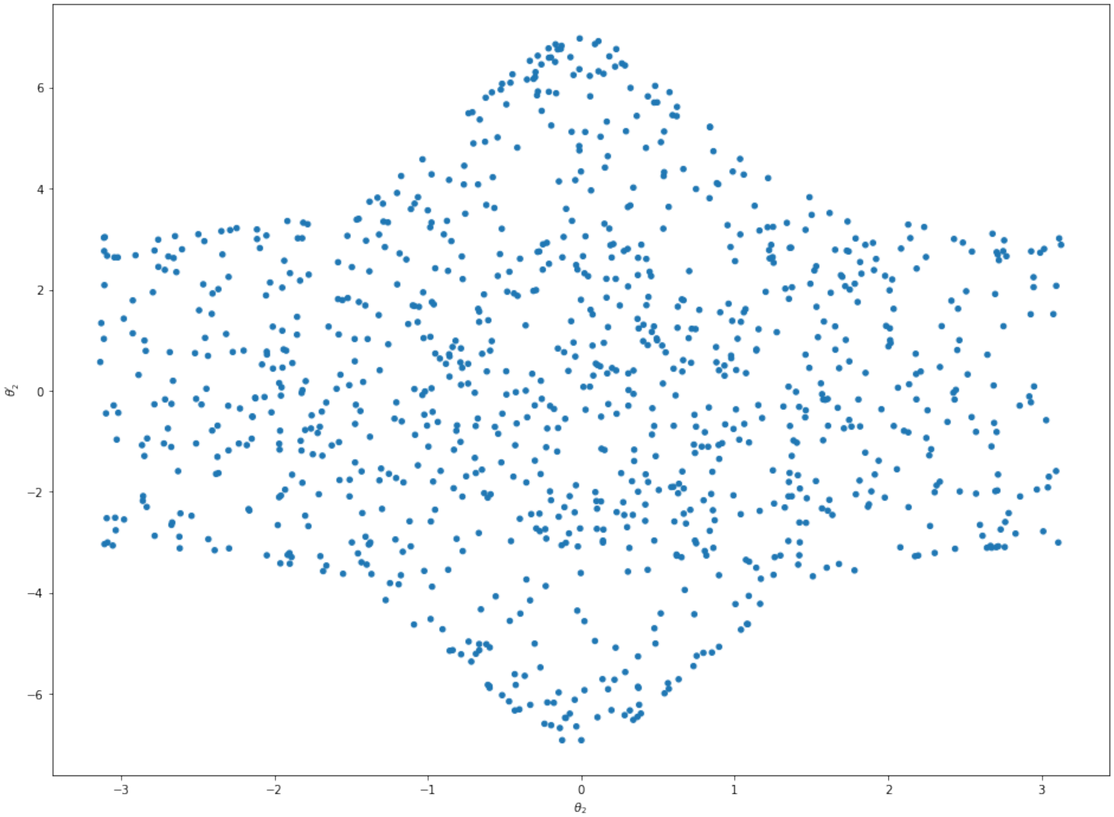

```
figsize(16,12)
plt.scatter(u0[2000:3000], v[2000:3000],
np.ones(len(u[1000:2000]))*20, marker="o")
plt.xlabel("$ \\theta_2 $")
plt.ylabel("$ \\theta_2' $")
plt.show()
```



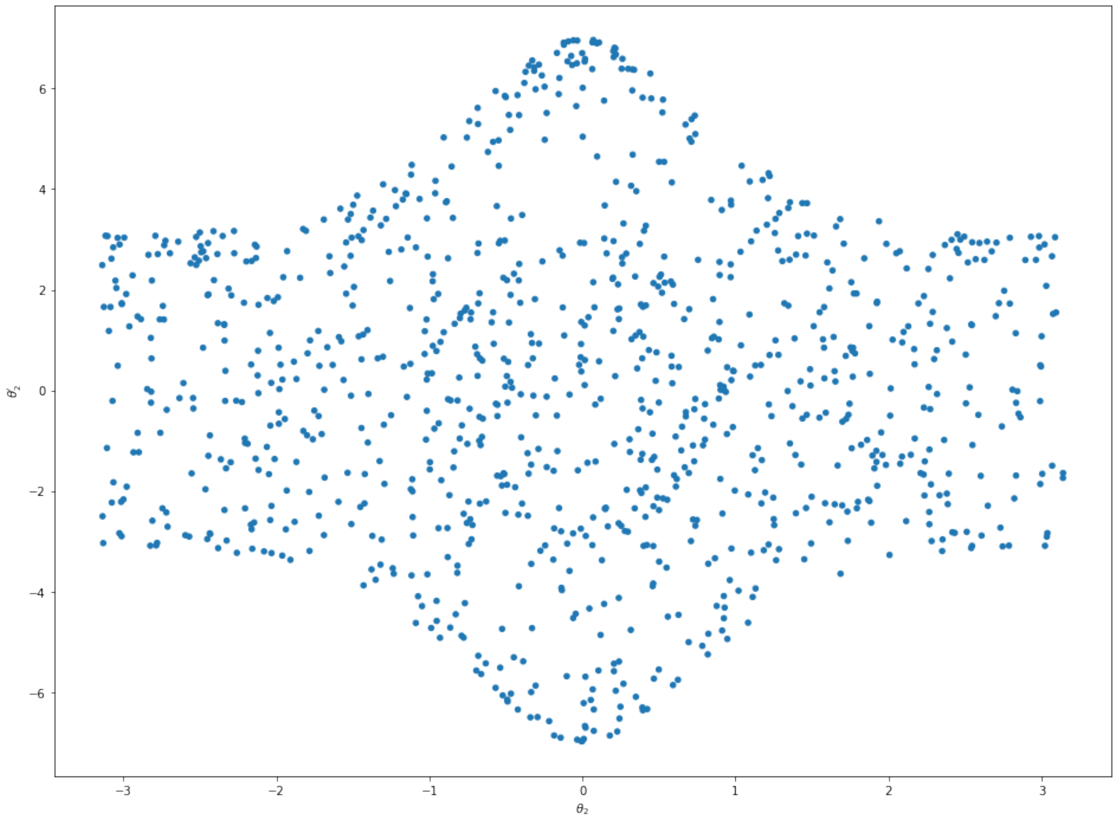

```
figsize(16,12)
plt.scatter(u0[3000:4000], v[3000:4000],
np.ones(len(u[1000:2000]))*20, marker="o")
plt.xlabel("$ \\theta_2 $")
plt.ylabel("$ \\theta_2' $")
plt.show()
```



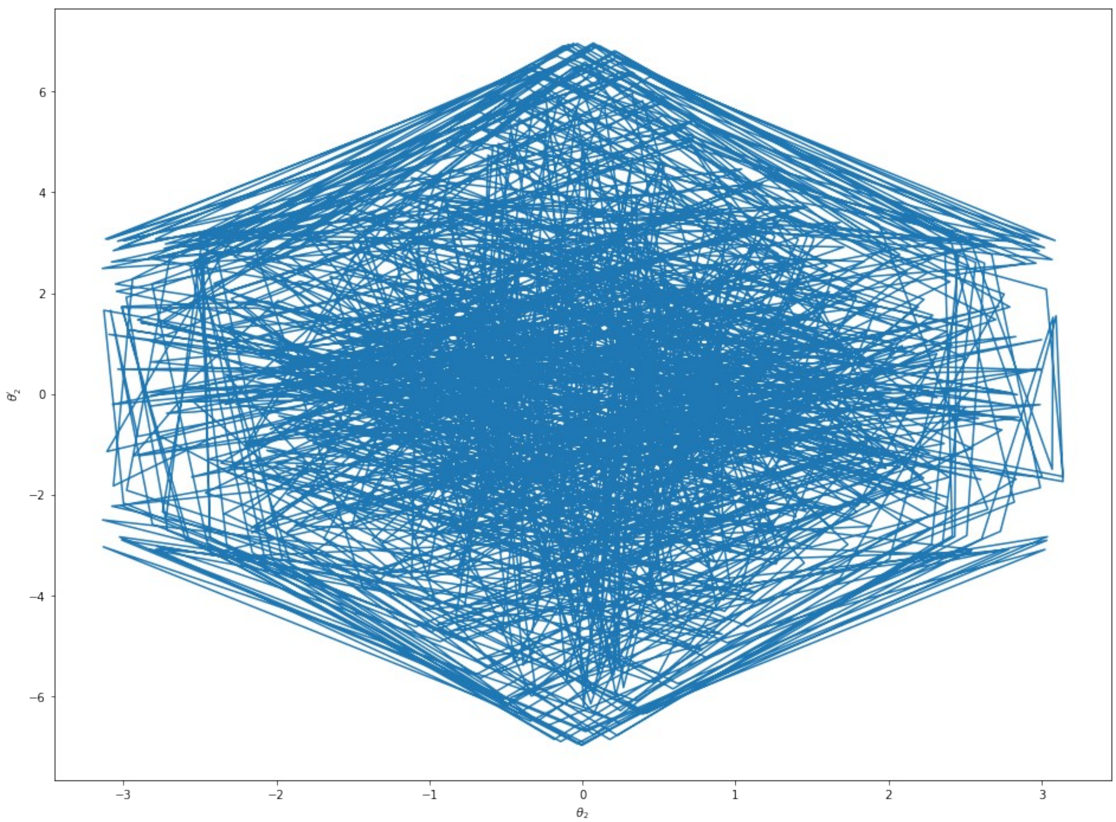

```
figsize(16,12)
plt.scatter(u0[4000:5000], v[4000:5000],
np.ones(len(u[1000:2000]))*20, marker="o")
plt.xlabel("$ \\theta_2 $")
plt.ylabel("$ \\theta_2' $")
plt.show()
```




```
figsize(16,12)
plt.scatter(u0[5000:6000], v[5000:6000],
np.ones(len(u[1000:2000]))*20, marker="o")
plt.xlabel("$ \\theta_2 $")
plt.ylabel("$ \\theta_2' $")
plt.show()
```




```
figsize(16,12)
plt.plot(u0[5000:6000], v[5000:6000])
plt.xlabel("$ \\theta_2 $")
plt.ylabel("$ \\theta_2' $")
plt.show()
```



Az első ábrán látható ellipszis kontúrján tömörülő pontok harmonikus oszcillációra utalnak. A haragnggröbére hasonlító burkoló görbe az előbbiekben említett sejthetően lezajló átlendülést írja le - $-\pi$-től a 0 felé (illetve π-től 0 felé) haladva θ_2-ben az abszolút szögsebesség nő, és $\theta_2 = 0$-ban globális maximuma van. Találhatóak ezeken kívül is zárt görbék, de nem annyira visszatérőek, hogy többezer másodperc alatt hasonló tömörülést mutassanak. Sejthető, hogy az ellipszisnek köze van a második inga sajátrezgézesihez. Ha az ellipszisen belül vagyunk, még akkor is lehetséges (ha a belső ingának nagy a szögsebessége), hogy a külső burkológörbén kötünk ki - hasonlóképp ha az ellipszisen kívül vagyunk, kiköthetünk az ellipszis belsejében. Az ellipszis szeparátrixnak nem nevezhető, a rendszer nem nevezhező stabilnak. Némi periódusosságot mutat, de inkább vannak (a rendszer fizikai mivolta miatt) egyes visszatérő mintázatok. A burkoló görbe is csak azért létezik, mert megmarad az energia, és megköti θ_2' maximumát. Még abban sincs semmi mintázat, hogy mikor áll meg a második inga - leszámítva az ellipszissel vett metszéspontokat. Az ellipszis és a burkoló görbe alatti sávok arra utalnak, hogy az energia csökken a numerikus integrálás miatt, ezért mindkettő idővel zsugorodik egy kicsit.

Kiegészíés

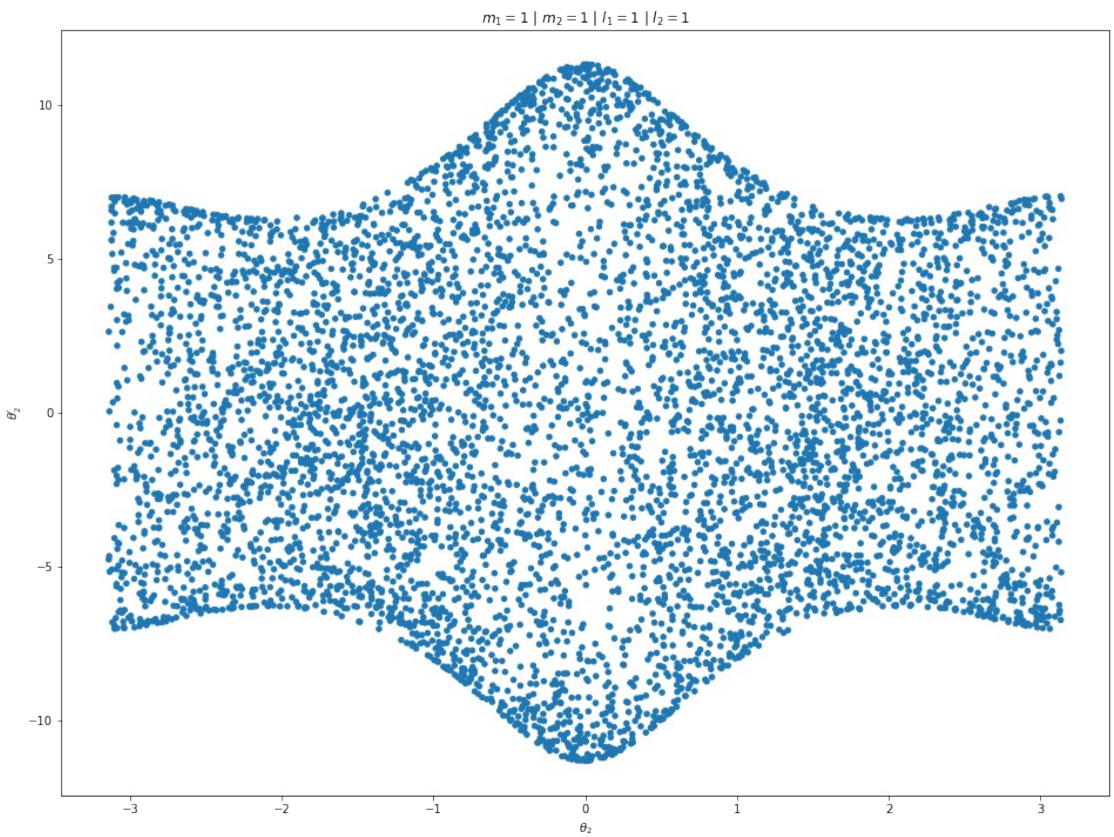
A fentiekben a két inga tömege megegyezett, az alsó inga kétszer olyan hosszú volt. Lehet variálni a tömegeket és a hosszokat tetszés szerint. Ezeket is exportálom .gif-be. Az elnevezések : "movei_m1m2l1l2.gif" ($m_1 = m_2 = l_1 = 1, l_2 = 2$: "movie1112.gif").

```
def kep_v(i):
    ax.plot([0, x1_v[i], x2_v[i]], [0, y1_v[i], y2_v[i]], lw=2, c='k')
# a két rúd
    c0 = Circle((0, 0), R/2, fc='k', zorder=10) # a felsfüggesztés
    c1 = Circle((x1_v[i], y1_v[i]), R, fc='b', ec='b', zorder=10) # az
első test
    c2 = Circle((x2_v[i], y2_v[i]), R, fc='r', ec='r', zorder=10) # a
második test
    ax.add_patch(c0)
    ax.add_patch(c1)
    ax.add_patch(c2)


    # a felfüggesztésre centráljuk a képet
    ax.set_xlim(-l1-l2-R, l1+l2+R)
    ax.set_ylim(-l1-l2-R, l1+l2+R)
    ax.set_aspect('equal', adjustable='box')
    plt.axis('off')
    plt.savefig("img_v{:d}.png".format(i//di), dpi=72)
    plt.cla()
```

$m_1=1, m_2=1, l_1=1, l_2=1$

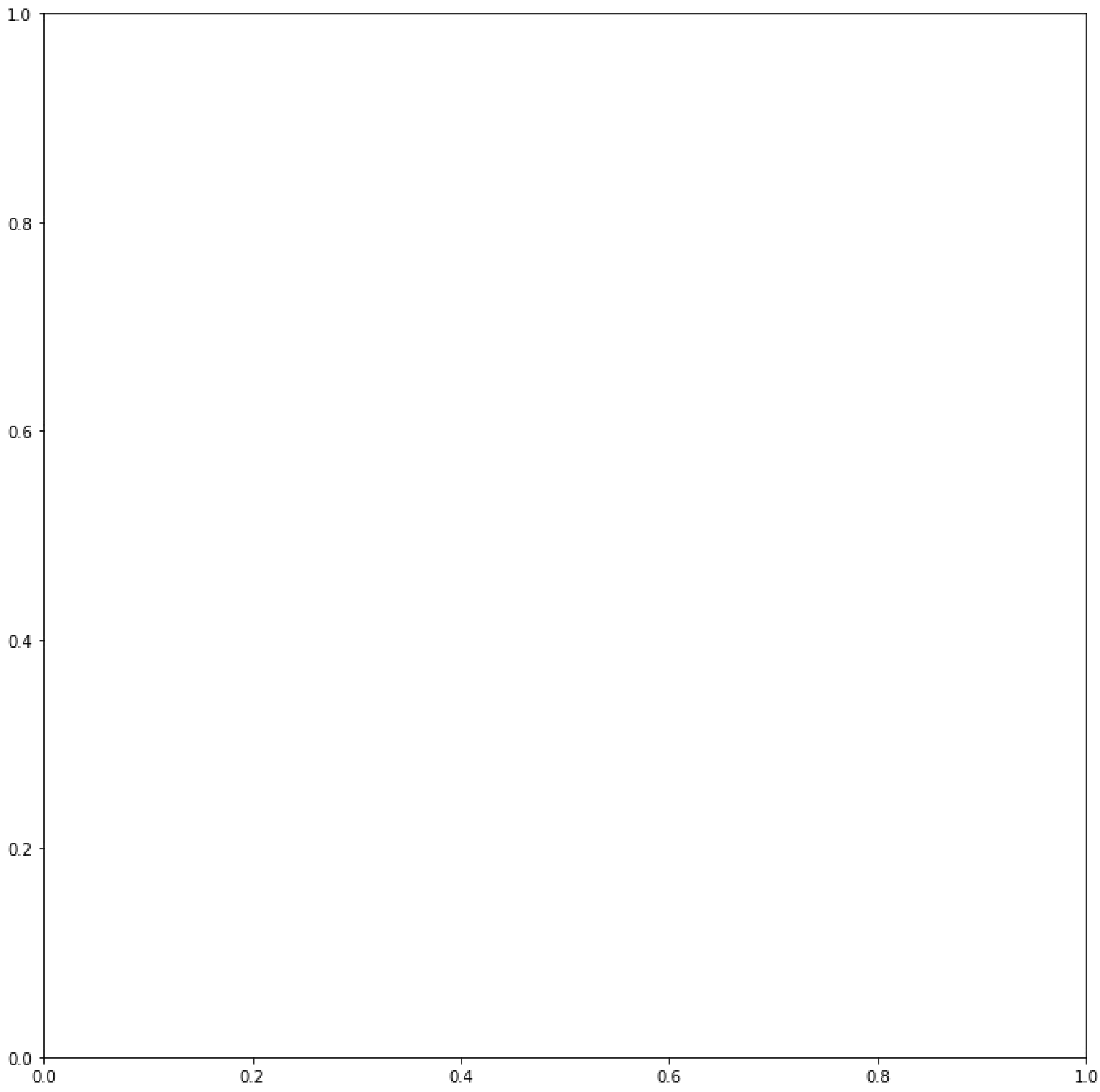
```
# Az ingák hosszai :
l1 = 1 # m
l2 = 1 # m
#Az ingák tömegei:
m1 = 1 # kg
m2 = 1 # kg

tmax, dt = 6000, 0.001 # meddig és milyen lépésközzel
t = np.arange(0, tmax+dt, dt) # mint a tömb
y0 = np.array([3*np.pi/7, 0, 3*np.pi/4, 0]) # kezdőfeltételek a dif
egyenletekhez
        # y alakja : th1, th1', th2, th2'

y = odeint(dif, y0, t, args=(l1, l2, m1, m2)) # az integrálás

th1, th2 = y[:,0], y[:,2]
dth1, dth2 = y[:,1], y[:,3]

u = [] # ide mennek a szögek
v = [] # ide mennek a szögsebességek

for i in range(len(th1)-1):
    buff1 = (th1[i] % pi)-pi*((th1[i]//pi)%2)
    buff2 = (th1[i+1] % pi)-pi*((th1[i+1]//pi)%2)
    if ((buff1<0) and (buff2>0)):
        u.append(th2[i])
        v.append(dth2[i])
    if ((buff2<0) and (buff1>0)):
```

```
        u.append(th2[i])
        v.append(dth2[i])

u0 = []
for i in range(len(u)):
    u0.append((u[i] % pi)-pi*((u[i]//pi)%2))

figsize(16,12)
plt.scatter(u0, v, np.ones(len(u))*20, marker="o")
tit = "$m_1 ="+str(m1)+"$ | $ m_2 =" + str(m2) + "$ | $ l_1 =" +
str(l1) + "$ | $ l_2 ="+ str(l2) +"$"
plt.title(tit)
plt.xlabel("$ \\theta_2 $")
plt.ylabel("$ \\theta_2' $")
plt.show()
```



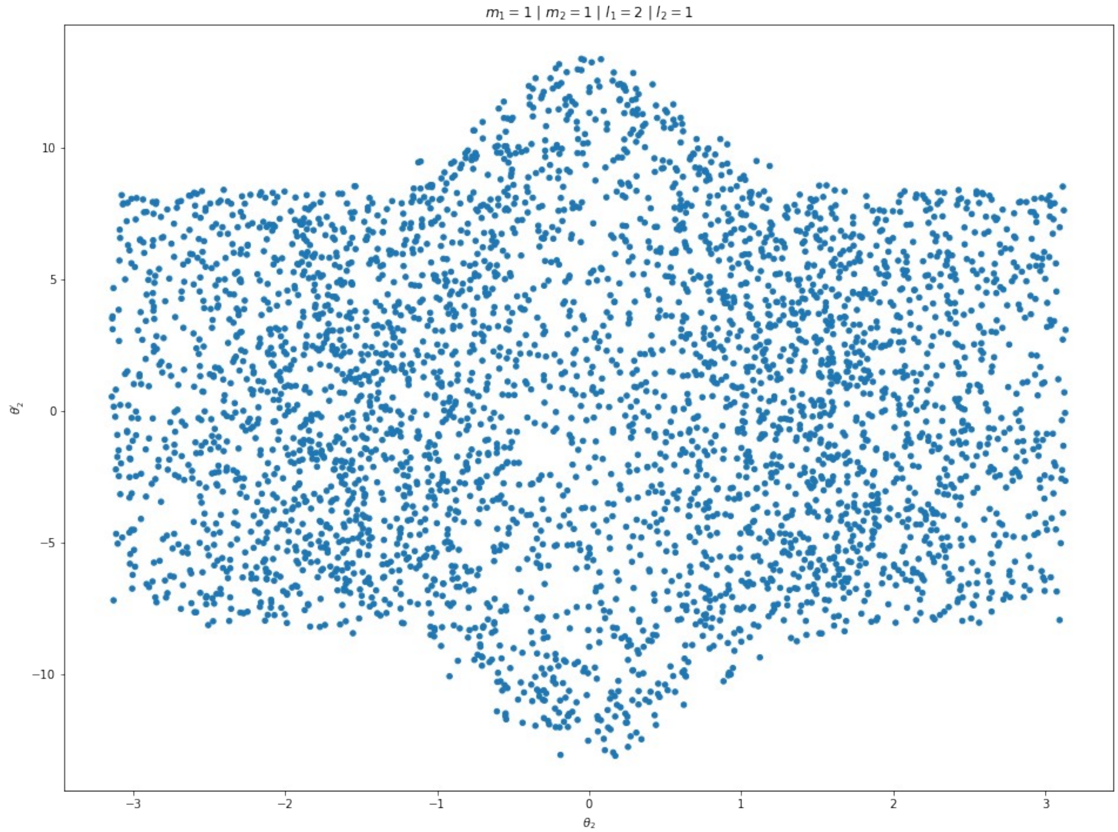

```
# Déscartes-koordináták
x1_v = l1 * np.sin(th1)
y1_v = -l1 * np.cos(th1)
x2_v = x1_v + l2 * np.sin(th2)
y2_v = y1_v - l2 * np.cos(th2)

# GIF
fps = 5
di = int(1/fps/dt)
```

```
fig = plt.figure(figsize=(16, 12), dpi=72)
ax = fig.add_subplot(111)

innen = 0
ennyit = 250*di

for i in range(innen, innen+ennyit, di):
    kep_v(i)

images = []
for i in range(innen//di, (innen+ennyit)//di):
    s = "img_v"+str(i)+".png"
    images.append(imageio.imread(s))
imageio.mimsave("movie1111.gif", images)
```

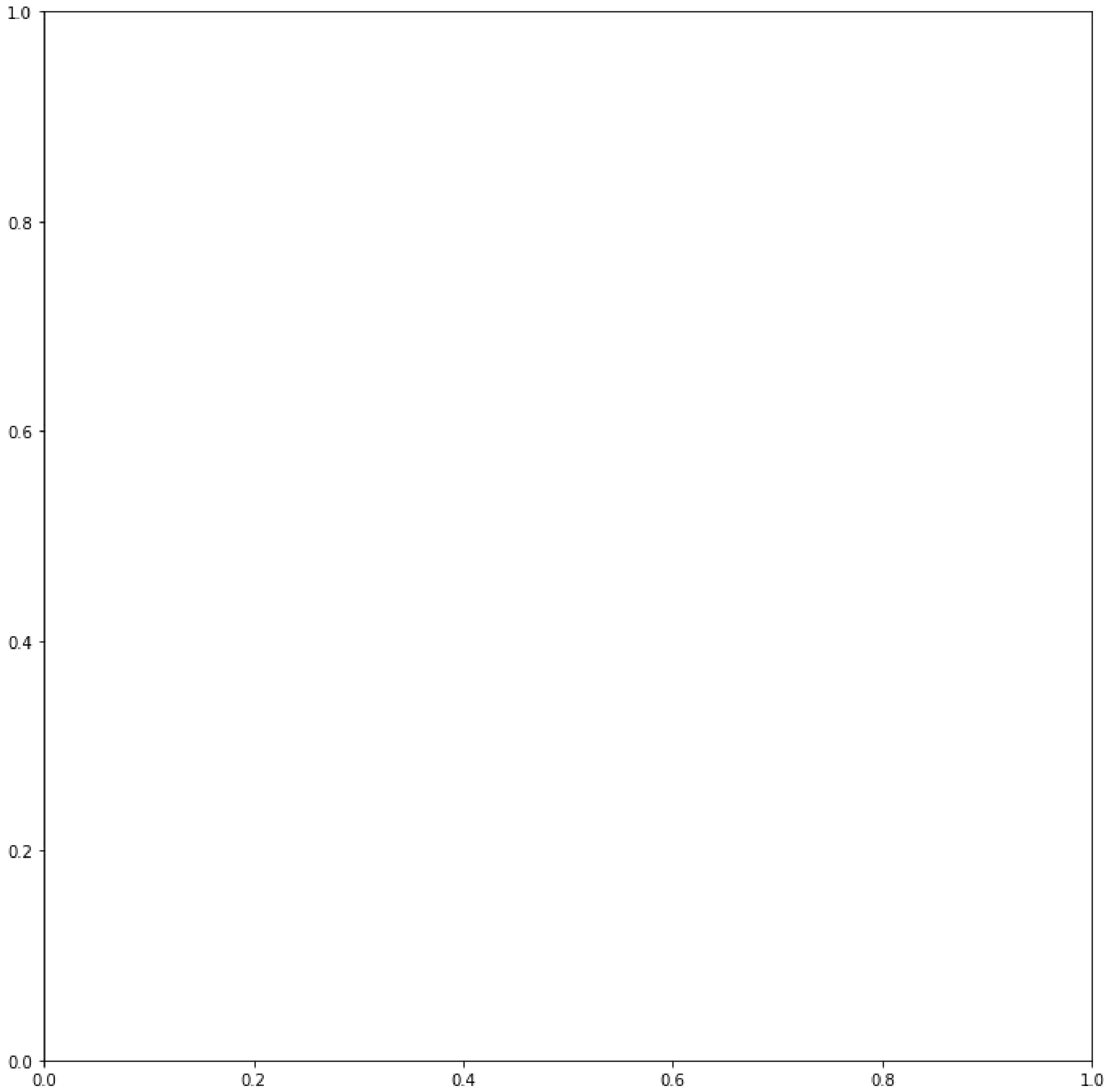


$m_1=1, m_2=1, l_1=2, l_2=1$

```
# Az ingák hosszai :
l1 = 2 # m
l2 = 1 # m
#Az ingák tömegei:
m1 = 1 # kg
m2 = 1 # kg

tmax, dt = 6000, 0.001 # meddig és milyen lépésközzel
t = np.arange(0, tmax+dt, dt) # mint a tömb
y0 = np.array([3*np.pi/7, 0, 3*np.pi/4, 0]) # kezdőfeltételek a dif
egyenletekhez
        # y alakja : th1, th1', th2, th2'

y = odeint(dif, y0, t, args=(l1, l2, m1, m2)) # az integrálás

th1, th2 = y[:,0], y[:,2]
dth1, dth2 = y[:,1], y[:,3]

u = [] # ide mennek a szögek
v = [] # ide mennek a szögsebességek

for i in range(len(th1_p)-1):
    buff1 = (th1[i] % pi)-pi*((th1[i]//pi)%2)
    buff2 = (th1[i+1] % pi)-pi*((th1[i+1]//pi)%2)
    if ((buff1<0) and (buff2>0)):
        u.append(th2[i])
        v.append(dth2[i])
    if ((buff2<0) and (buff1>0)):
        u.append(th2[i])
        v.append(dth2[i])

u0 = []
for i in range(len(u)):
    u0.append((u[i] % pi)-pi*((u[i]//pi)%2))

figsize(16,12)
plt.scatter(u0, v, np.ones(len(u))*20, marker="o")
tit = "$m_1 ="+str(m1)+"$ | $ m_2 =" + str(m2) + "$ | $ l_1 =" +
str(l1) + "$ | $ l_2 ="+ str(l2) +"$"
plt.title(tit)
plt.xlabel("$ \\theta_2 $")
plt.ylabel("$ \\theta_2' $")
plt.show()
```



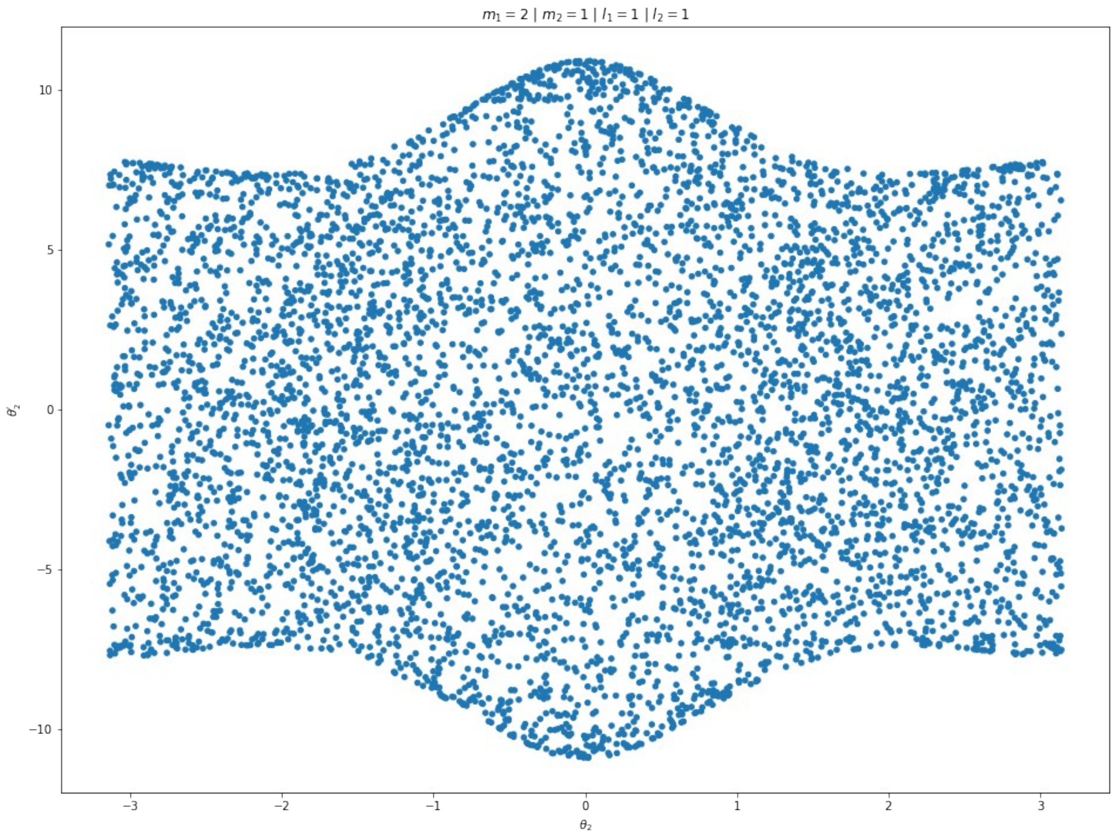

```python
# Déscartes-koordináták
x1_v = l1 * np.sin(th1)
y1_v = -l1 * np.cos(th1)
x2_v = x1_v + l2 * np.sin(th2)
y2_v = y1_v - l2 * np.cos(th2)

# GIF
fps = 5
di = int(1/fps/dt)
fig = plt.figure(figsize=(16, 12), dpi=72)
ax = fig.add_subplot(111)

innen = 0
ennyit = 250*di

for i in range(innen, innen+ennyit, di):
    kep_v(i)

images = []
for i in range(innen//di, (innen+ennyit)//di):
    s = "img_v"+str(i)+".png"
    images.append(imageio.imread(s))
imageio.mimsave("movie1121.gif", images)
```



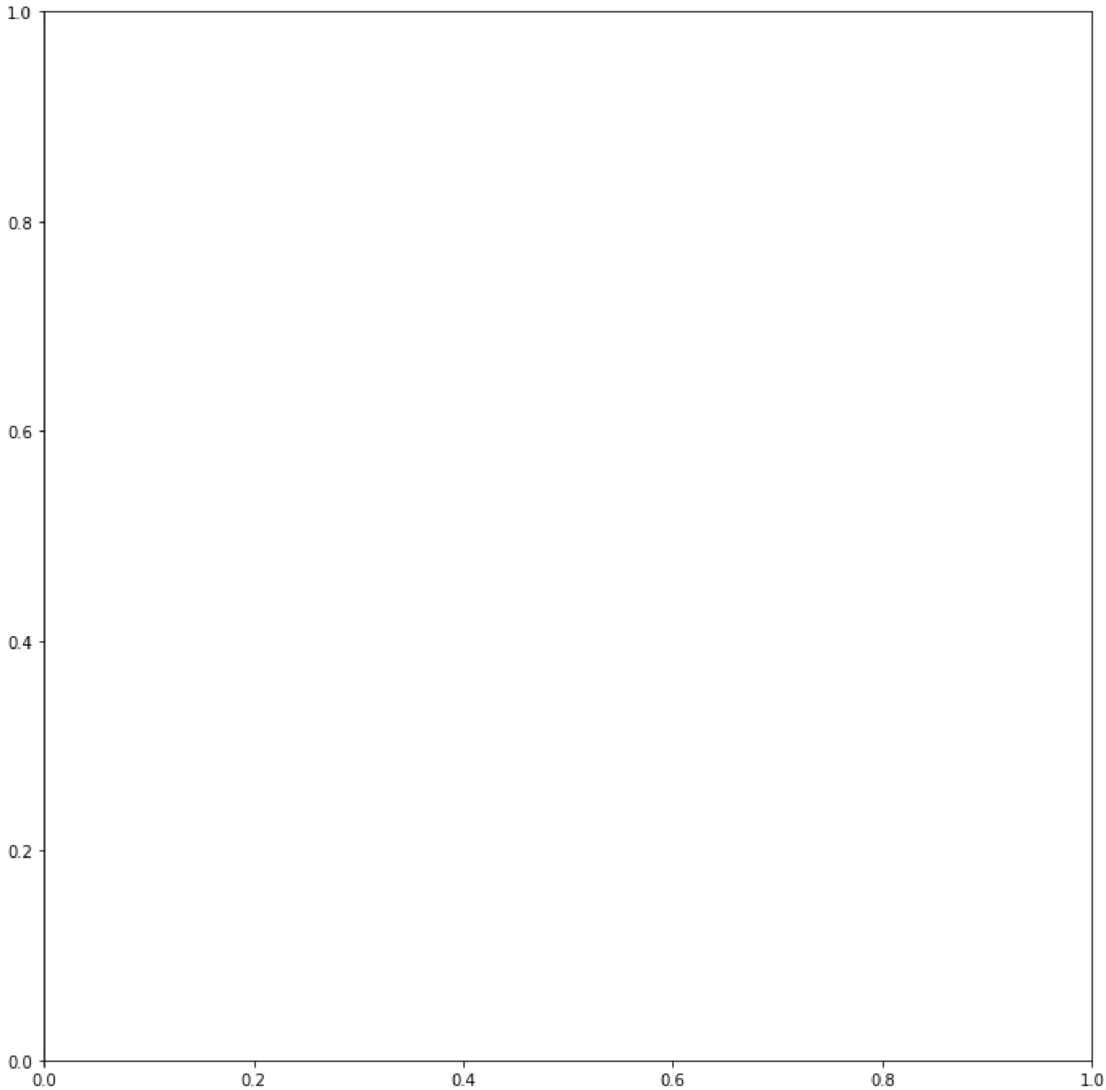
$m_1=2, m_2=1, l_1=1, l_2=1$

```
# Az ingák hosszai :
l1 = 1 # m
l2 = 1 # m
#Az ingák tömegei:
m1 = 2 # kg
m2 = 1 # kg

tmax, dt = 6000, 0.001 # meddig és milyen lépésközzel
t = np.arange(0, tmax+dt, dt) # mint a tömb
y0 = np.array([3*np.pi/7, 0, 3*np.pi/4, 0]) # kezdőfeltételek a dif
egyenletekhez
        # y alakja : th1, th1', th2, th2'

y = odeint(dif, y0, t, args=(l1, l2, m1, m2)) # az integrálás

th1, th2 = y[:,0], y[:,2]
```

```python
dth1, dth2 = y[:,1], y[:,3]

u = [] # ide mennek a szögek
v = [] # ide mennek a szögsebességek

for i in range(len(th1_p)-1):
    buff1 = (th1[i] % pi)-pi*((th1[i]//pi)%2)
    buff2 = (th1[i+1] % pi)-pi*((th1[i+1]//pi)%2)
    if ((buff1<0) and (buff2>0)):
        u.append(th2[i])
        v.append(dth2[i])
    if ((buff2<0) and (buff1>0)):
        u.append(th2[i])
        v.append(dth2[i])

u0 = []
for i in range(len(u)):
    u0.append((u[i] % pi)-pi*((u[i]//pi)%2))

figsize(16,12)
plt.scatter(u0, v, np.ones(len(u))*20, marker="o")
tit = "$m_1 ="+str(m1)+"$ | $ m_2 =" + str(m2) + "$ | $ l_1 =" +
str(l1) + "$ | $ l_2 ="+ str(l2) +"$"
plt.title(tit)
plt.xlabel("$ \\theta_2 $")
plt.ylabel("$ \\theta_2' $")
plt.show()
```



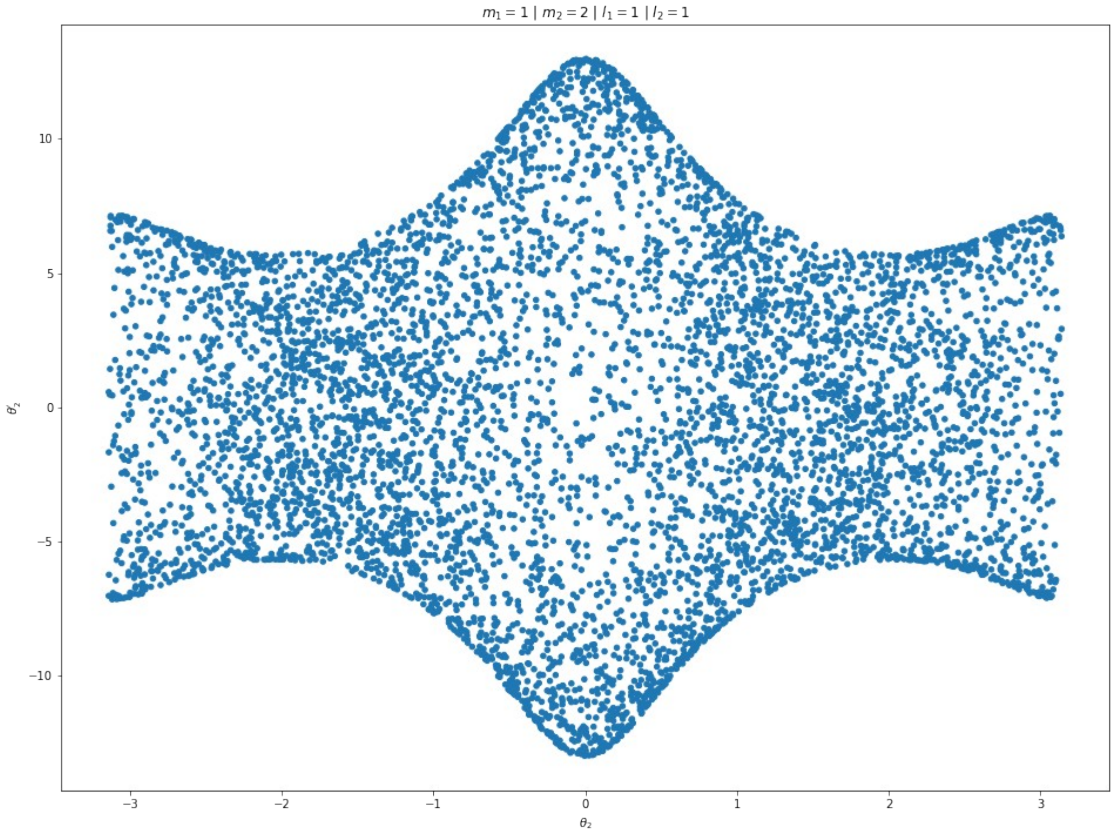

```
# Déscartes-koordináták
x1_v = l1 * np.sin(th1)
y1_v = -l1 * np.cos(th1)
x2_v = x1_v + l2 * np.sin(th2)
y2_v = y1_v - l2 * np.cos(th2)

# GIF
fps = 5
di = int(1/fps/dt)
fig = plt.figure(figsize=(16, 12), dpi=72)
ax = fig.add_subplot(111)

innen = 0
ennyit = 250*di

for i in range(innen, innen+ennyit, di):
    kep_v(i)

images = []
for i in range(innen//di, (innen+ennyit)//di):
    s = "img_v"+str(i)+".png"
    images.append(imageio.imread(s))
imageio.mimsave("movie2111.gif", images)
```



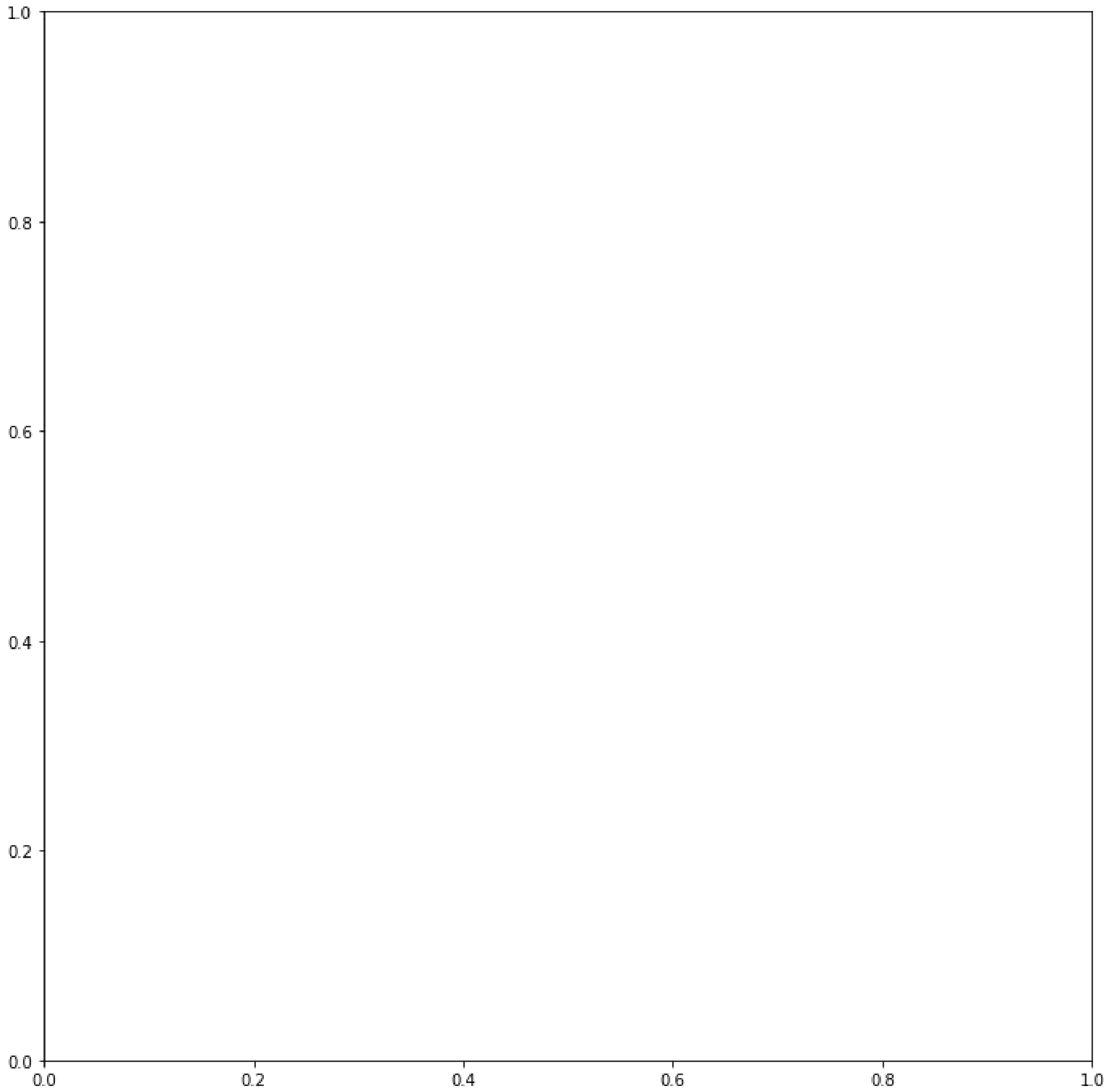
$m_1=1, m_2=2, l_1=1, l_2=1$

```
# Az ingák hosszai :
l1 = 1 # m
l2 = 1 # m
#Az ingák tömegei:
m1 = 1 # kg
m2 = 2 # kg

tmax, dt = 6000, 0.001 # meddig és milyen lépésközzel
t = np.arange(0, tmax+dt, dt) # mint a tömb
y0 = np.array([3*np.pi/7, 0, 3*np.pi/4, 0]) # kezdőfeltételek a dif
egyenletekhez
        # y alakja : th1, th1', th2, th2'

y = odeint(dif, y0, t, args=(l1, l2, m1, m2)) # az integrálás

th1, th2 = y[:,0], y[:,2]
```

```python
dth1, dth2 = y[:,1], y[:,3]

u = [] # ide mennek a szögek
v = [] # ide mennek a szögsebességek

for i in range(len(th1_p)-1):
    buff1 = (th1[i] % pi)-pi*((th1[i]//pi)%2)
    buff2 = (th1[i+1] % pi)-pi*((th1[i+1]//pi)%2)
    if ((buff1<0) and (buff2>0)):
        u.append(th2[i])
        v.append(dth2[i])
    if ((buff2<0) and (buff1>0)):
        u.append(th2[i])
        v.append(dth2[i])

u0 = []
for i in range(len(u)):
    u0.append((u[i] % pi)-pi*((u[i]//pi)%2))

figsize(16,12)
plt.scatter(u0, v, np.ones(len(u))*20, marker="o")
tit = "$m_1 ="+str(m1)+"$ | $ m_2 =" + str(m2) + "$ | $ l_1 =" +
str(l1) + "$ | $ l_2 ="+ str(l2) +"$"
plt.title(tit)
plt.xlabel("$ \\theta_2 $")
plt.ylabel("$ \\theta_2' $")
plt.show()
```



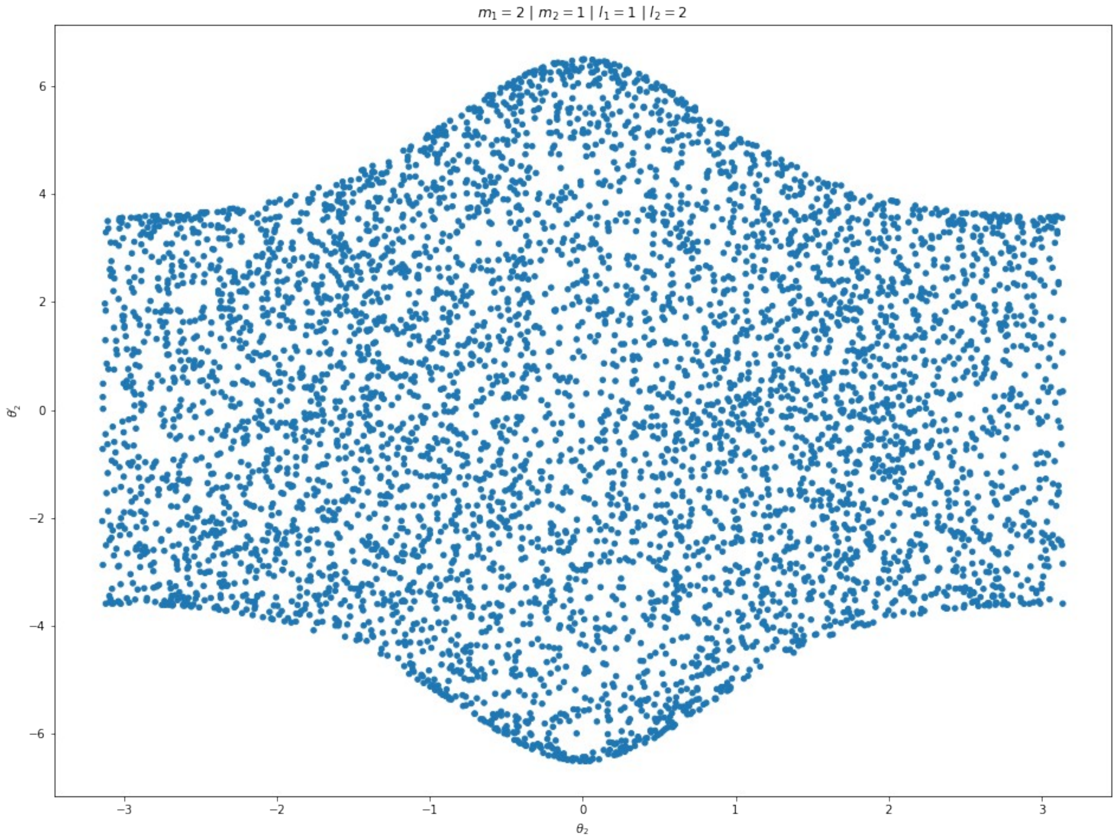

```
# Déscartes-koordináták
x1_v = l1 * np.sin(th1)
y1_v = -l1 * np.cos(th1)
x2_v = x1_v + l2 * np.sin(th2)
y2_v = y1_v - l2 * np.cos(th2)

# GIF
fps = 5
di = int(1/fps/dt)
fig = plt.figure(figsize=(16, 12), dpi=72)
ax = fig.add_subplot(111)

innen = 0
ennyit = 250*di

for i in range(innen, innen+ennyit, di):
    kep_v(i)

images = []
for i in range(innen//di, (innen+ennyit)//di):
    s = "img_v"+str(i)+".png"
    images.append(imageio.imread(s))
imageio.mimsave("movie1211.gif", images)
```



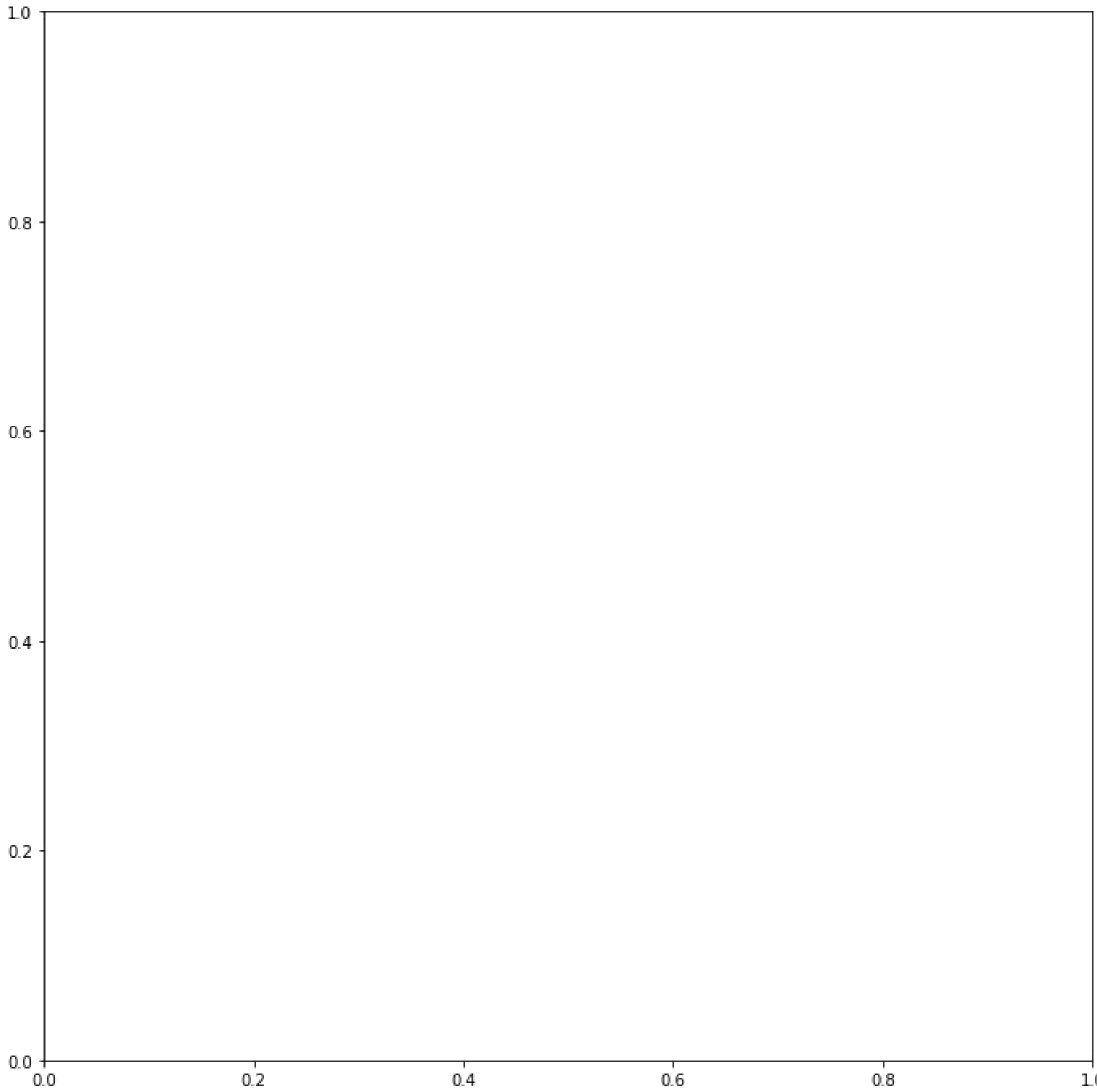
$m_1=2, m_2=1, l_1=1, l_2=2$

```
# Az ingák hosszai :
l1 = 1 # m
l2 = 2 # m
#Az ingák tömegei:
m1 = 2 # kg
m2 = 1 # kg

tmax, dt = 6000, 0.001 # meddig és milyen lépésközzel
t = np.arange(0, tmax+dt, dt) # mint a tömb
y0 = np.array([3*np.pi/7, 0, 3*np.pi/4, 0]) # kezdőfeltételek a dif
egyenletekhez
        # y alakja : th1, th1', th2, th2'

y = odeint(dif, y0, t, args=(l1, l2, m1, m2)) # az integrálás

th1, th2 = y[:,0], y[:,2]
```

```python
dth1, dth2 = y[:,1], y[:,3]

u = [] # ide mennek a szögek
v = [] # ide mennek a szögsebességek

for i in range(len(th1_p)-1):
    buff1 = (th1[i] % pi)-pi*((th1[i]//pi)%2)
    buff2 = (th1[i+1] % pi)-pi*((th1[i+1]//pi)%2)
    if ((buff1<0) and (buff2>0)):
        u.append(th2[i])
        v.append(dth2[i])
    if ((buff2<0) and (buff1>0)):
        u.append(th2[i])
        v.append(dth2[i])

u0 = []
for i in range(len(u)):
    u0.append((u[i] % pi)-pi*((u[i]//pi)%2))

figsize(16,12)
plt.scatter(u0, v, np.ones(len(u))*20, marker="o")
tit = "$m_1 ="+str(m1)+"$ | $ m_2 =" + str(m2) + "$ | $ l_1 =" +
str(l1) + "$ | $ l_2 ="+ str(l2) +"$"
plt.title(tit)
plt.xlabel("$ \\theta_2 $")
plt.ylabel("$ \\theta_2' $")
plt.show()
```



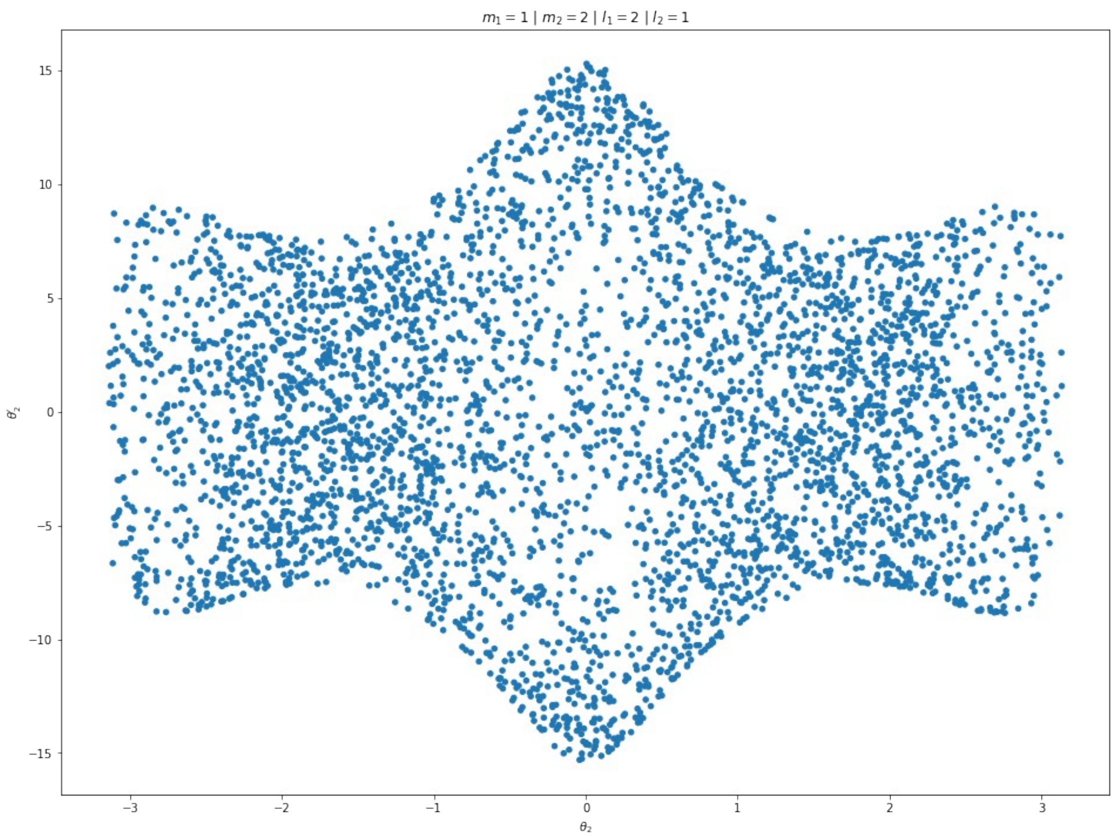

```python
# Déscartes-koordináták
x1_v = l1 * np.sin(th1)
y1_v = -l1 * np.cos(th1)
x2_v = x1_v + l2 * np.sin(th2)
y2_v = y1_v - l2 * np.cos(th2)

# GIF
fps = 5
di = int(1/fps/dt)
fig = plt.figure(figsize=(16, 12), dpi=72)
ax = fig.add_subplot(111)

innen = 0
ennyit = 250*di

for i in range(innen, innen+ennyit, di):
    kep_v(i)

images = []
for i in range(innen//di, (innen+ennyit)//di):
    s = "img_v"+str(i)+".png"
    images.append(imageio.imread(s))
imageio.mimsave("movie2112.gif", images)
```



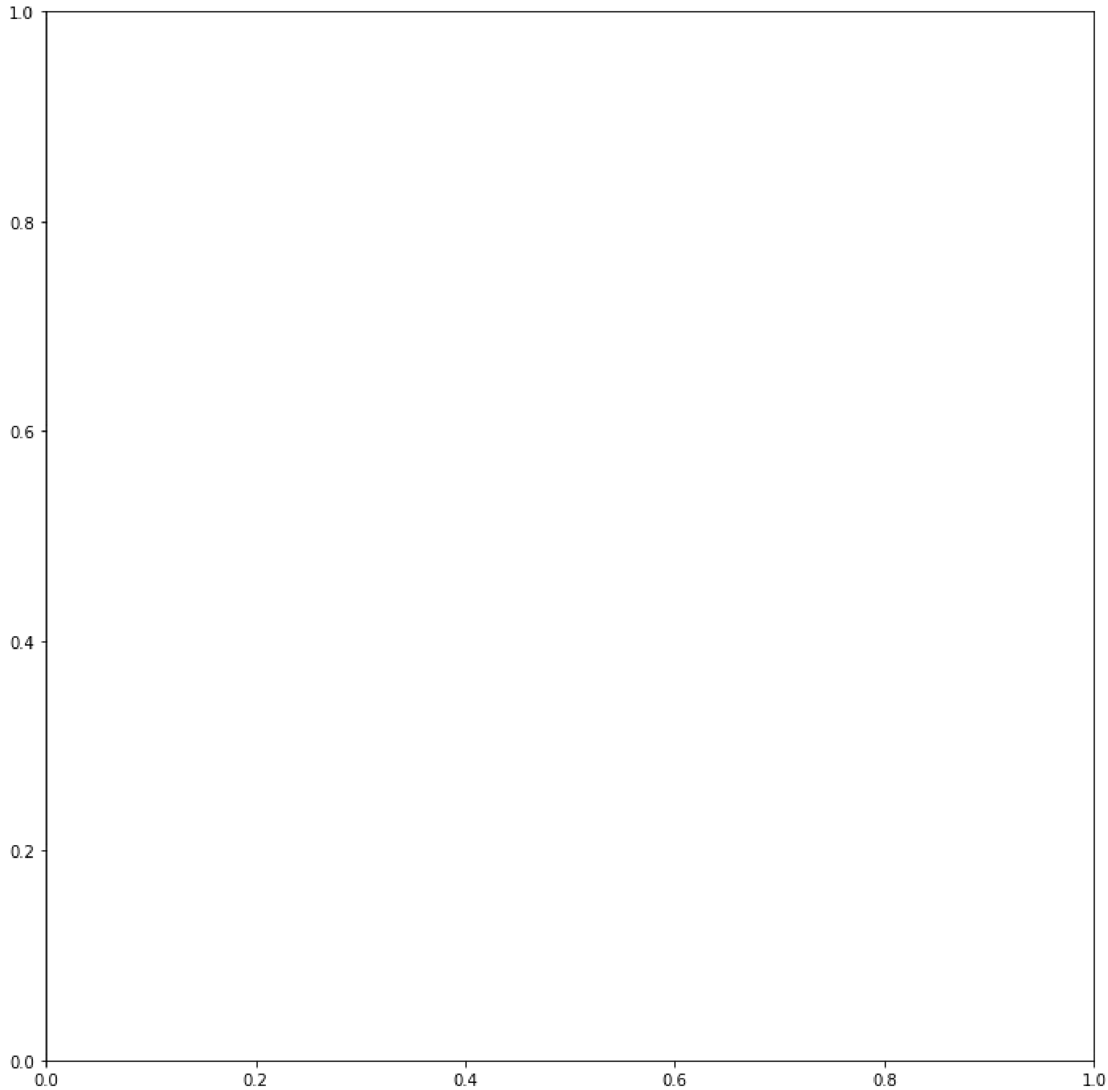
$m_1=1, m_2=2, l_1=2, l_2=1$

```
# Az ingák hosszai :
l1 = 2 # m
l2 = 1 # m
#Az ingák tömegei:
m1 = 1 # kg
m2 = 2 # kg

tmax, dt = 6000, 0.001 # meddig és milyen lépésközzel
t = np.arange(0, tmax+dt, dt) # mint a tömb
y0 = np.array([3*np.pi/7, 0, 3*np.pi/4, 0]) # kezdőfeltételek a dif
egyenletekhez
        # y alakja : th1, th1', th2, th2'

y = odeint(dif, y0, t, args=(l1, l2, m1, m2)) # az integrálás

th1, th2 = y[:,0], y[:,2]
```

```python
dth1, dth2 = y[:,1], y[:,3]

u = [] # ide mennek a szögek
v = [] # ide mennek a szögsebességek

for i in range(len(th1_p)-1):
    buff1 = (th1[i] % pi)-pi*((th1[i]//pi)%2)
    buff2 = (th1[i+1] % pi)-pi*((th1[i+1]//pi)%2)
    if ((buff1<0) and (buff2>0)):
        u.append(th2[i])
        v.append(dth2[i])
    if ((buff2<0) and (buff1>0)):
        u.append(th2[i])
        v.append(dth2[i])

u0 = []
for i in range(len(u)):
    u0.append((u[i] % pi)-pi*((u[i]//pi)%2))

figsize(16,12)
plt.scatter(u0, v, np.ones(len(u))*20, marker="o")
tit = "$m_1 ="+str(m1)+"$ | $ m_2 =" + str(m2) + "$ | $ l_1 =" +
str(l1) + "$ | $ l_2 ="+ str(l2) +"$"
plt.title(tit)
plt.xlabel("$ \\theta_2 $")
plt.ylabel("$ \\theta_2' $")
plt.show()
```




```python
# Déscartes-koordináták
x1_v = l1 * np.sin(th1)
y1_v = -l1 * np.cos(th1)
x2_v = x1_v + l2 * np.sin(th2)
y2_v = y1_v - l2 * np.cos(th2)

# GIF
fps = 5
di = int(1/fps/dt)
fig = plt.figure(figsize=(16, 12), dpi=72)
ax = fig.add_subplot(111)

innen = 0
ennyit = 250*di

for i in range(innen, innen+ennyit, di):
    kep_v(i)

images = []
for i in range(innen//di, (innen+ennyit)//di):
    s = "img_v"+str(i)+".png"
    images.append(imageio.imread(s))
imageio.mimsave("movie1221.gif", images)
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0