Función perturbadora en Python con Celmech

una alternativa a DisturbingFunction.m

Elisa Castro Martínez

`celmech` [1] es un nuevo paquete de Python que ofrece diversas funcionalidades en relación a la mecánica celeste. En particular, permite trabajar directamente con la formulación analítica de la función perturbadora.

En este trabajo se desarrolla un módulo basado en las funciones de `celmech` que calcula y muestra la función perturbadora en términos de coordenadas heliocéntricas no canónicas, para una partícula de masa despreciable perturbada por un objeto de masa m con órbita fija. El objetivo es generar un recurso alternativo al código `DisturbingFunction.m` para el entorno Mathematica desarrollado por Ellis y Murray [2], pero en Python.

Hay dos motivos por los que `celmech` no es totalmente equivalente. Primero, no permite el uso de partículas de prueba (sin masa), ya que siempre considera las perturbaciones mutuas entre los objetos, hallando el hamiltoniano del sistema y no de la partícula, mientras que `DisturbingFunction` considera la partícula perturbada de masa despreciable y el objeto perturbador en órbita fija; y segundo, `celmech` utiliza un sistema de coordenadas diferente a `DisturbingFunction`, introduciendo una diferencia importante ya que `celmech` no distingue entre perturbaciones internas y externas.

Cálculo de $\mathcal{R}_I$ y $\mathcal{R}_E$

Siguiendo el desarrollo descrito en Murray y Dermott (2000) [3] y Ellis y Murray (2000) [2] la función perturbadora puede escribirse como la suma de un término directo y uno indirecto, este último siendo diferente en el caso de un perturbador externo ($\mathcal{R}_E$) y un perturbador interno ($\mathcal{R}_I$). En `celmech` no se realiza esta distinción debido a que se utilizan variables canónicas, mientras que el sistema astrocéntrico usado en este trabajo es no canónico. Como resultado, la parte directa de la función perturbadora (que solamente depende de la distancia entre los objetos) es igual en ambos sistemas, pero la parte indirecta (que depende del sistema elegido) no. Por lo tanto, el principal objetivo de este trabajo es calcular la parte indirecta de la función perturbadora en el sistema astrocéntrico.

Lo hacemos mirando las ecuaciones 65 y 66 de Ellis y Murray (2000):

$$\begin{aligned}\mathcal{R}_E = -\kappa_m \frac{(1-m)!}{(1+m)!} \times F_{1,m,p}(i) F_{1,m,p'}(i') X_{-j_2}^{1,-j_2-j_4}(e) X_{j_1}^{-2,j_1+j_3}(e') \\ \times \cos(j_1\lambda' + j_2\lambda + j_3\varpi' + j_4\varpi + j_5\Omega' + j_6\Omega)\end{aligned}$$

$$\begin{aligned}\mathcal{R}_I = -\kappa_m \frac{(1-m)!}{(1+m)!} \times F_{1,m,p}(i) F_{1,m,p'}(i') X_{-j_2}^{-2,-j_2-j_4}(e) X_{j_1}^{1,j_1+j_3}(e') \\ \times \cos(j_1\lambda' + j_2\lambda + j_3\varpi' + j_4\varpi + j_5\Omega' + j_6\Omega)\end{aligned}$$

donde las funciones $F_{l,m,p}(i)$ son funciones de la inclinación de Kaula y $X_c^{a,b}(e)$ son los coeficientes de Hansen, y los parámetros m, p, p' y κ_m se definen a partir de $j_1, j_2, j_3, j_4, j_5, j_6$. Los términos primados corresponden al objeto exterior.

`celmech` ofrece todas las funciones necesarias, solamente hay que definir los coeficientes. Esto se realizó en `func_pert.coef_E` y `func_pert.coef_I` para el caso externo e interno respectivamente. Estas funciones calculan los coeficientes (todo lo que está antes del coseno en las ecuaciones anteriores) dados $j_1, j_2, j_3, j_4, j_5, j_6$ y el orden de s, s', e y e'.

Función perturbadora

Utilizando las funciones `list_secular_terms` y `list_resonance_terms` del módulo `celmech.disturbing_function` podemos encontrar los términos relevantes hasta cierto orden de la función perturbadora secular y resonante respectivamente ($\mathcal{R}_E$, $\mathcal{R}_I$ y $\mathcal{R}_D$). A partir de ello, hallando los coeficientes de dichos términos con las funciones anteriores, podemos hallar la función perturbadora correspondiente usando `sympy`, según las ecuaciones:

$$\mathcal{R} = \frac{\mu'}{a'}\mathcal{R}_D + \frac{\mu'}{a'}\alpha\mathcal{R}_E$$
$$\mathcal{R}' = \frac{\mu}{a'}\mathcal{R}_D + \frac{\mu}{a'}\frac{1}{\alpha^2}\mathcal{R}_I$$

donde $\alpha = a/a'$.

La función `funcion_perturbadora` del módulo `func_pert` toma como argumentos la salida de `list_secular_terms` y `list_resonance_terms`, los elementos orbitales (más $\mu = k^2 m$) del cuerpo interior y exterior (pudiendo ser valores numéricos y/o objetos simbólicos de `sympy`), el tipo de perturbación (interna o externa) y el valor del parámetro α (puede ser un valor numérico entre 0 y 1 o vacío, en cuyo caso no se computará el valor numérico y se obtendrá una expresión en función de α). La ventaja de utilizar `sympy` es que es muy fácil transformar la función obtenida en formato LaTeX, es cómodo para visualizar y manipular algebraicamente y también se puede convertir fácilmente en funciones numéricas para integrar numéricamente las ecuaciones de movimiento usando, por ejemplo, `scipy`.

Ejemplo 1: caso genérico secular

Primero importamos los paquetes necesarios.

```
import numpy as np
import sympy as sy

from celmech import disturbing_function as df

import func_pert as fp

from scipy.integrate import odeint, solve_ivp

import matplotlib.pyplot as plt
import matplotlib
plt.rcParams['text.usetex'] = True
sy.init_printing()

matplotlib.rc('xtick', labelsize=8)
matplotlib.rc('ytick', labelsize=8)
plt.rcParams.update({'font.size': 8})
```

Generamos la lista con los términos relevantes en la función perturbadora hasta orden 2 en inclinaciones y excentricidades.

```
A_sec = df.list_secular_terms(0,2)

A_sec
```

$$((0,\ 0,\ 0,\ 0,\ 0,\ 0),\ (0,\ 0,\ 0,\ 0)),\ ((0,\ 0,\ 0,\ 0,\ 0,\ 0),\ (0,\ 0,\ 0,\ 1)),$$
$$((0,\ 0,\ 0,\ 0,\ 0,\ 0),\ (0,\ 0,\ 1,\ 0)),\ ((0,\ 0,\ 0,\ 0,\ 0,\ 0),\ (0,\ 1,\ 0,\ 0)),$$
$$((0,\ 0,\ 0,\ 0,\ 0,\ 0),\ (1,\ 0,\ 0,\ 0)),\ ((0,\ 0,\ -1,\ 1,\ 0,\ 0),\ (0,\ 0,\ 0,\ 0)),$$
$$((0,\ 0,\ 0,\ 0,\ -1,\ 1),\ (0,\ 0,\ 0,\ 0))$$

Para escribir la función perturbadora, definimos los elementos orbitales (más μ) para cada objeto como símbolos `sympy` $(a, e, i, \varpi, \Omega, \lambda, \mu)$ para el objeto interior y exterior. En este ejemplo los subíndices i corresponden al objeto interior y los j al exterior. También definimos un vector `elementos` que contiene los elementos orbitales en orden, para el objeto interior y exterior.

```
a_i,a_j,e_i,e_j,i_i,i_j,varpi_i,varpi_j,Omega_i,Omega_j,lambda_i,lambda_j,
mu_i,mu_j = sy.symbols('a_i,a_j,e_i,e_j,i_i,i_j,varpi_i,varpi_j,Omega_i,
                       Omega_j,lambda_i,lambda_j,mu_i,mu_j')

elementos = [a_i,e_i,i_i,varpi_i,Omega_i,lambda_i,mu_i,a_j,e_j,i_j,varpi_j,
             Omega_j,lambda_j,mu_j]

#Escribimos la función perturbadora SECULAR
R_sec_E = fp.funcion_perturbadora(A_sec,*elementos,tipo='e')

#Visualizamos
R_sec_E
```

$$\frac{C^{(0,0,0,0)}_{(0,0,-1,1,0,0)}(\alpha_{i,j})e_ie_j\mu_j\cos\left(\varpi_i-\varpi_j\right)}{a_j}+\frac{C^{(0,0,0,0)}_{(0,0,0,0,-1,1)}(\alpha_{i,j})\mu_j\sin\left(\frac{i_i}{2}\right)\sin\left(\frac{i_j}{2}\right)\cos\left(\Omega_i-\Omega_j\right)}{a_j}+\frac{C^{(0,0,0,0)}_{(0,0,0,0,0,0)}(\alpha_{i,j})\mu_j}{a_j}+\frac{C^{(0,0,0,1)}_{(0,0,0,0,0,0)}(\alpha_{i,j})e_j^2\mu_j}{a_j}+\frac{C^{(0,0,1,0)}_{(0,0,0,0,0,0)}(\alpha_{i,j})e_i^2\mu_j}{a_j}+\frac{C^{(0,1,0,0)}_{(0,0,0,0,0,0)}(\alpha_{i,j})\mu_j\sin^2\left(\frac{i_j}{2}\right)}{a_j}+\frac{C^{(1,0,0,0)}_{(0,0,0,0,0,0)}(\alpha_{i,j})\mu_j\sin^2\left(\frac{i_i}{2}\right)}{a_j}$$

Los coeficientes $C^{(\nu_1,\nu_2,\nu_3,\nu_4)}_{(j_1,j_2,j_3,j_4,j_5,j_6)}$ corresponden a los coeficientes de la parte directa de la función perturbadora, que dependen únicamente de α. Aparecerán cuando no introduzcamos un valor para el argumento `Alpha` en `fp.funcion_perturbadora`.

Si quisiéramos que el objeto perturbado fuera el interior, simplemente cambiamos el tipo de función perturbadora:

```
R_sec_I = fp.funcion_perturbadora(A_sec,*elementos,tipo='i')

#Visualizamos
R_sec_I
```

$$\frac{C^{(0,0,0,0)}_{(0,0,-1,1,0,0)}(\alpha_{i,j})e_ie_j\mu_i\cos\left(\varpi_i-\varpi_j\right)}{a_j}+\frac{C^{(0,0,0,0)}_{(0,0,0,0,-1,1)}(\alpha_{i,j})\mu_i\sin\left(\frac{i_i}{2}\right)\sin\left(\frac{i_j}{2}\right)\cos\left(\Omega_i-\Omega_j\right)}{a_j}+\frac{C^{(0,0,0,0)}_{(0,0,0,0,0,0)}(\alpha_{i,j})\mu_i}{a_j}+\frac{C^{(0,0,0,1)}_{(0,0,0,0,0,0)}(\alpha_{i,j})e_j^2\mu_i}{a_j}+\frac{C^{(0,0,1,0)}_{(0,0,0,0,0,0)}(\alpha_{i,j})e_i^2\mu_i}{a_j}+\frac{C^{(0,1,0,0)}_{(0,0,0,0,0,0)}(\alpha_{i,j})\mu_i\sin^2\left(\frac{i_j}{2}\right)}{a_j}+\frac{C^{(1,0,0,0)}_{(0,0,0,0,0,0)}(\alpha_{i,j})\mu_i\sin^2\left(\frac{i_i}{2}\right)}{a_j}$$

Ejemplo 2: asteroide perturbado por Júpiter (secular)

Veamos una aplicación concreta, basada en el ejemplo 6.9.1 de Murray y Dermott. Consideramos un asteroide afectado por Júpiter, lejos de cualquier resonancia de movimientos medios. Elegimos un semieje para el asteroide con $a_{i0} = 0{,}998976$ UA, por lo que $\alpha = 0{,}192$. También definimos los elementos de Júpiter ya que asumimos que son constantes en el tiempo.

```
a_i,a_j,e_i,e_j,i_i,i_j,varpi_i,varpi_j,Omega_i,Omega_j,lambda_i,lambda_j,
mu_i,mu_j = sy.symbols('a_i,a_j,e_i,e_j,i_i,i_j,varpi_i,varpi_j,Omega_i,
                       Omega_j,lambda_i,lambda_j,mu_i,mu_j')

# Constante gravitacional en días, masas solares y ua
k = 0.01720209895

# Determinamos las características de la órbita del objeto perturbador
#(que consideramos fija)
mu_j =  k**2 * 1/1047.355
varpi_j = 0
Omega_j = 0
e_j = 0.048
i_j = np.deg2rad(1.035)
a_j = 5.203

a_i0 = 0.192*a_j
alpha = a_i0/a_j

elementos = [a_i,e_i,i_i,varpi_i,Omega_i,lambda_i,mu_i,a_j,e_j,i_j,varpi_j,
             Omega_j,lambda_j,mu_j]

R_sec_E = fp.funcion_perturbadora(A_sec,*elementos,tipo='e',Alpha=alpha)
```

$$\mathcal{R}_{sec} = 8{,}05486771802467 \cdot 10^{-10} e_i^2 - 1{,}84718980999374 \cdot 10^{-11} e_i \cos(\varpi_i) - \\ -3{,}22194708720987 \cdot 10^{-9} \sin^2\left(\frac{i_i}{2}\right) + 5{,}82009691400805 \cdot 10^{-11} \sin\left(\frac{i_i}{2}\right) \cos(\Omega_i) + \\ 5{,}48146052442939 \cdot 10^{-8}$$

Para encontrar la evolución planteamos las ecuaciones planetarias de Lagrange. Primero diferenciamos respecto a los elementos:

```
DRa = sy.diff(R_sec_E,a_i)
DRa
```

0

```
DRvarpi = sy.diff(R_sec_E,varpi_i)
DRvarpi
```

$1{,}84718980999374 \cdot 10^{-11} e_i \sin(\varpi_i)$

```
DROmega = sy.diff(R_sec_E,Omega_i)
DROmega
```

$-5{,}82009691400805 \cdot 10^{-11} \sin(\Omega_i) \sin\left(\frac{i_i}{2}\right)$

```
DRe = sy.diff(R_sec_E,e_i)
DRe
```

$$1{,}61097354360493 \cdot 10^{-9} e_i - 1{,}84718980999374 \cdot 10^{-11} \cos\left(\varpi_i\right)$$

```
DRi = sy.diff(R_sec_E,i_i)
DRi
```

$$-3{,}22194708720987 \cdot 10^{-9} \sin\left(\frac{i_i}{2}\right) \cos\left(\frac{i_i}{2}\right) + 2{,}91004845700402 \cdot 10^{-11} \cos\left(\Omega_i\right) \cos\left(\frac{i_i}{2}\right)$$

```
DRlambda_o = sy.diff(R_sec_E,lambda_i)
DRlambda_o
```

Escribimos el sistema:

```
n = k/(a_i**(3/2))

apunto_s = 2/(n*a_i) * DRlambda_o

epunto_s = -(1-e_i**2)**(1/2)/(n*a_i**2*e_i) * DRvarpi -
    (1-e_i**2)**(1/2)/(n*a_i**2*e_i) * (1-(1-e_i**2)**(1/2) ) * DRlambda_o

Omegapunto_s = 1/(n*a_i**2*(1-e_i**2)**(1/2)*sy.sin(i_i))*DRi

ipunto_s = -sy.tan(i_i/2) / (n*a_i**2*(1-e_i**2)**(1/
↪2))*(DRvarpi+DRlambda_o)-
        1/(n*a_i**2*(1-e_i**2)**(1/2)*sy.sin(i_i))*DROmega

varpipunto_s = (1-e_i**2)**(1/2)/(n*a_i**2*e_i)*DRe +
        sy.tan(i_i/2)/(n*a_i**2*(1-e_i**2)**(1/2))*DRi

lambdaopunto_s = -(1-e_i**2)/(n*a_i**2*e_i)*DRe -  2/(n*a_i)*DRa+ n +␣
↪varpipunto_s

lambdappunto_s = k/a_j**(3/2)
```

`lambdaopunto` corresponde a la derivada temporal de la longitud media del asteroide, y `lambdappunto` a la derivada temporal de la longitud media de Júpiter.

En el desarrollo anterior se incluyeron las ecuaciones triviales para $\frac{d\lambda_p}{dt}$ y $\frac{da}{dt}$. Esto es porque, cuando se estudie el caso con resonancias, será necesario incluirlas en el sistema. Se mantuvieron para tener que introducir la mínima cantidad de cambios cuando trabajemos con uno u otro caso.

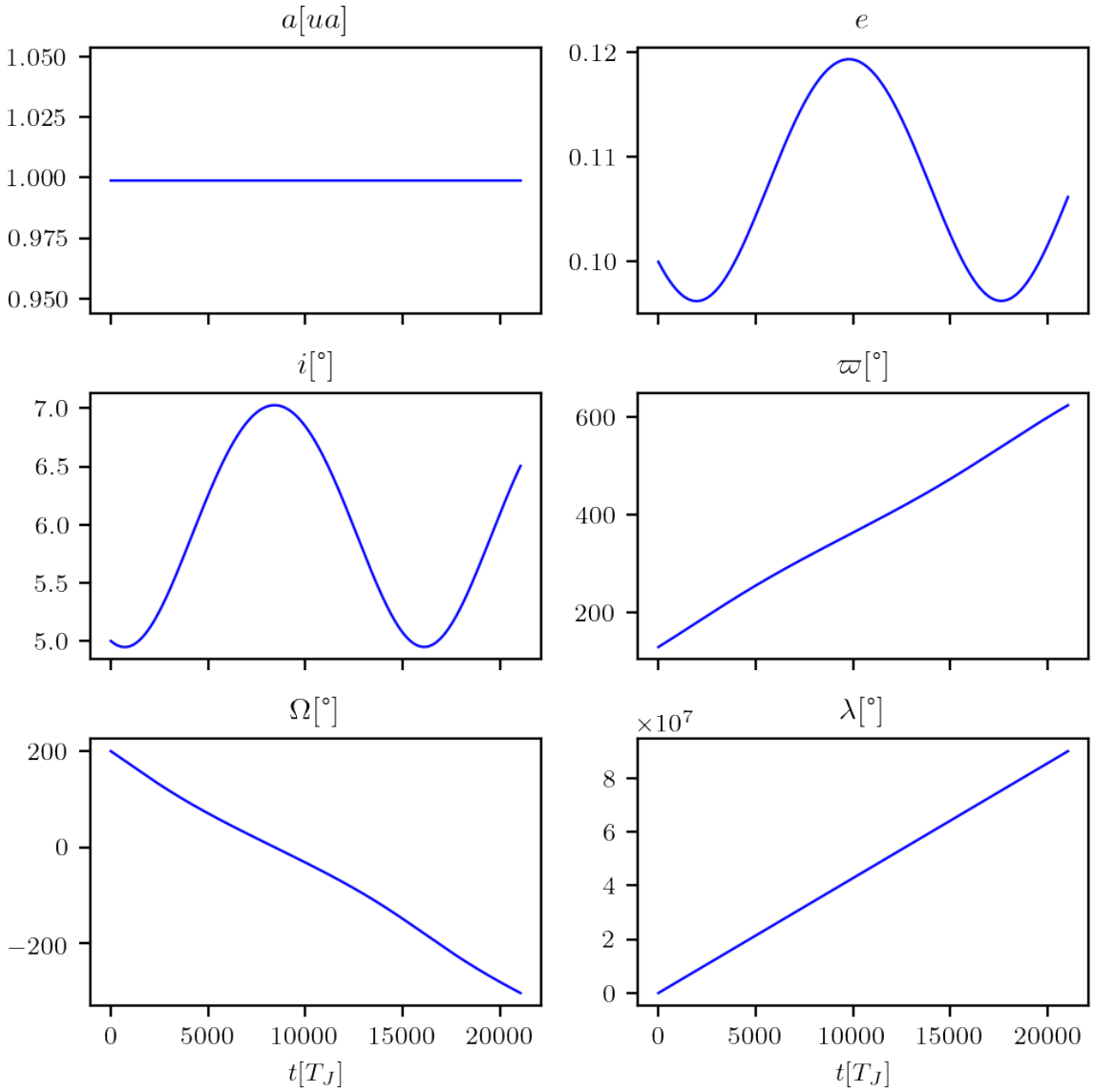
Usando `scipy` podemos integrar el sistema, pero antes debemos transformar los objetos simbólicos de `sympy` a funciones, utilizando `sympy.lambdify`.

```
apunto_f=sy.lambdify([a_i,e_i,i_i,varpi_i,Omega_i,lambda_i,lambda_j],
                            apunto_s)
epunto_f = sy.lambdify([a_i,e_i,i_i,varpi_i,Omega_i,lambda_i,lambda_j],
                            epunto_s)
varpipunto_f = sy.lambdify([a_i,e_i,i_i,varpi_i,Omega_i,lambda_i,lambda_j],
                            varpipunto_s)
Omegapunto_f = sy.lambdify([a_i,e_i,i_i,varpi_i,Omega_i,lambda_i,lambda_j],
                            Omegapunto_s)
ipunto_f = sy.lambdify([a_i,e_i,i_i,varpi_i,Omega_i,lambda_i,lambda_j],
                            ipunto_s)
lambdaipunto_f = sy.lambdify([a_i,e_i,i_i,varpi_i,Omega_i,lambda_i,lambda_j],
                            lambdaopunto_s)
lambdajpunto_f = sy.lambdify([a_i,e_i,i_i,varpi_i,Omega_i,lambda_i,lambda_j],
                            lambdappunto_s)
```

Ahora sí, usando `scipy.integrate` integramos las ecuaciones, para ciertas condiciones iniciales y_0:

```
def lagrange_sist(t, variables):
    a_i,e_i,i_i,varpi_i,Omega_i,lambda_i,lambda_j = variables

    apunto = apunto_f(a_i,e_i,i_i,varpi_i,Omega_i,lambda_i,lambda_j)
    epunto = epunto_f(a_i,e_i,i_i,varpi_i,Omega_i,lambda_i,lambda_j)
    ipunto = ipunto_f(a_i,e_i,i_i,varpi_i,Omega_i,lambda_i,lambda_j)
    varpipunto = varpipunto_f(a_i,e_i,i_i,varpi_i,Omega_i,lambda_i,lambda_j)
    Omegapunto = Omegapunto_f(a_i,e_i,i_i,varpi_i,Omega_i,lambda_i,lambda_j)
    lambdaipunto =␣
 ↪lambdaipunto_f(a_i,e_i,i_i,varpi_i,Omega_i,lambda_i,lambda_j)
    lambdajpunto =␣
 ↪lambdajpunto_f(a_i,e_i,i_i,varpi_i,Omega_i,lambda_i,lambda_j)

    return␣
 ↪[apunto,epunto,ipunto,varpipunto,Omegapunto,lambdaipunto,lambdajpunto]

#Condiciones iniciales y0 =␣
 ↪(a(0),e(0),i(0),varpi(0),Omega(0),lambda_i(0),lambda_j(0))
y0 = [a_i0,0.1,np.deg2rad(5),np.deg2rad(130),np.deg2rad(200),np.
 ↪deg2rad(300),0]

#t = arange(tiempo_inicial, tiempo_final[días], paso_temporal)
t = np.arange(0, 365*250000, 100000)

#Solución
evolucion = odeint(lagrange_sist, y0, t, tfirst=True)
```

Graficando los resultados:


$a[ua]$
1.050
1.025
1.000
0.975
0.950
e
0.12
0.11
0.10
$i[^\circ]$
7.0
6.5
6.0
5.5
5.0
$\varpi[^\circ]$
600
400
200
$\Omega[^\circ]$
200
0
−200
0
5000
10000
15000
20000
$t[T_J]$
$\lambda[^\circ]$
$\times 10^7$
8
6
4
2
0
0
5000
10000
15000
20000
$t[T_J]$

Ejemplo 3: Transneptuniano perturbado por Neptuno (secular)

Por completitud, se incluye un ejemplo para un perturbador interno. Usamos los datos correspondientes a un transneptuniano perturbado por Neptuno.

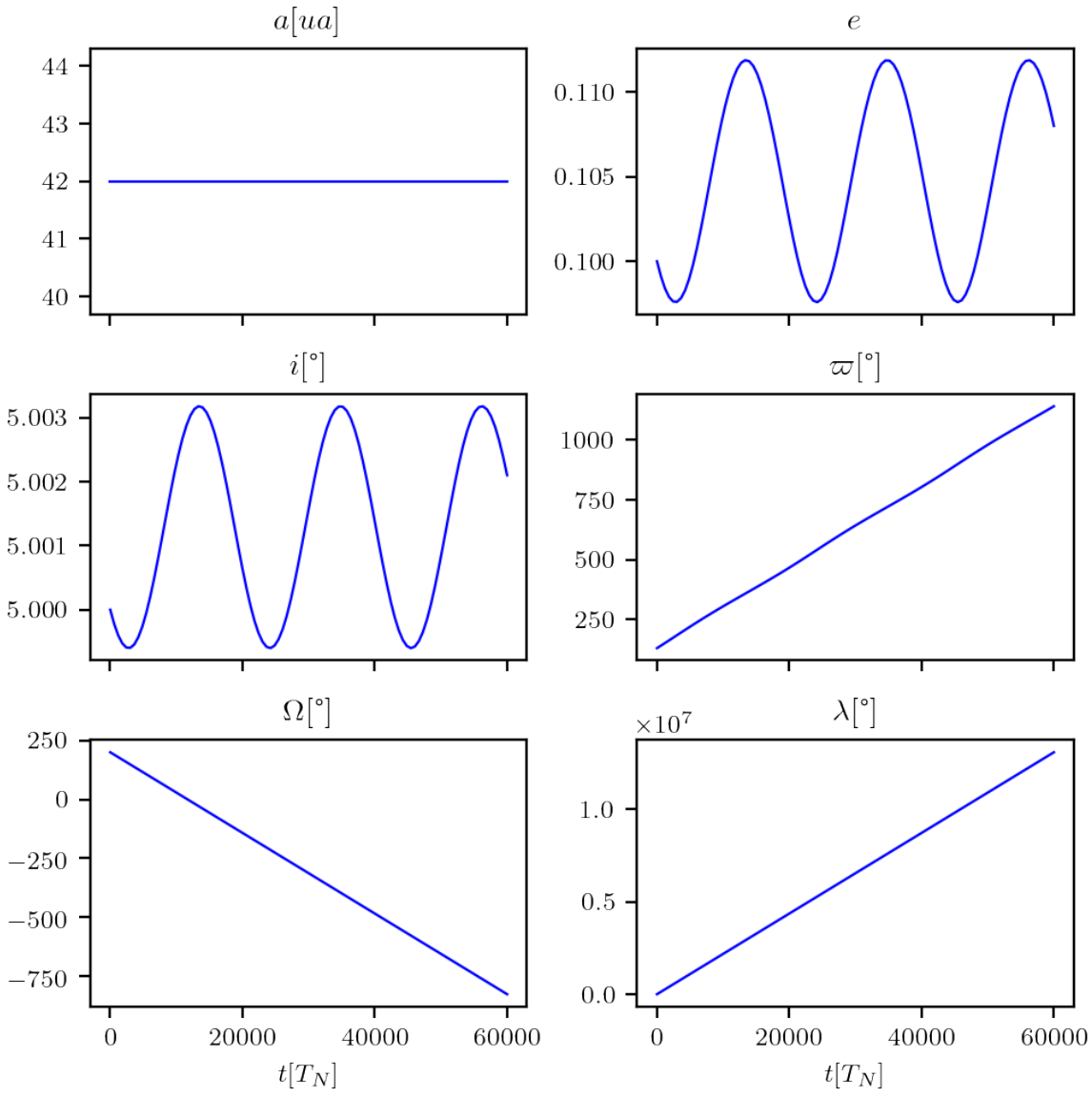
```
a_N,a_j,e_N,e_j,i_N,i_j,varpi_N,varpi_j,Omega_N,Omega_j,lambda_N,lambda_j,
mu_N,mu_j=sy.symbols('a_N,a_j,e_N,e_j,i_N,i_j,varpi_N,varpi_j,
                Omega_N,Omega_j,lambda_N,lambda_j,mu_N,mu_j')

mu_N =  k**2 * 0.00005149
varpi_N = 0
Omega_N = 0
e_N = 0.00859
i_N = 0
a_N = 30.1
a_j0 = 42
alpha = a_N/a_j0

elementos=[a_N,e_N,i_N,varpi_N,Omega_N,lambda_N,mu_N,a_j,e_j,i_j,varpi_j,
        Omega_j,lambda_j,mu_j]

R_sec_I = fp.funcion_perturbadora(A_sec,*elementos,tipo='i',Alpha=alpha)

[DRa,DRe,DRi,DRvarpi,DROmega,DRlambda_o] = [sy.diff(R_sec_I,a_j),
                                            sy.diff(R_sec_I,e_j),
                                            sy.diff(R_sec_I,i_j),
                                            sy.diff(R_sec_I,varpi_j),
                                            sy.diff(R_sec_I,Omega_j),
                                            sy.diff(R_sec_I,lambda_j)]
```

$$\mathcal{R}_{sec} = \frac{1{,}15463402468577 \cdot 10^{-8} e_j^2}{a_j} - \frac{1{,}63957268379591 \cdot 10^{-10} e_j \cos\left(\varpi_j\right)}{a_j} - \\ - \frac{4{,}61853609874309 \cdot 10^{-8} \sin^2\left(\frac{i_j}{2}\right)}{a_j} + \frac{1{,}80986864633801 \cdot 10^{-8}}{a_j}$$

Planteamos las ecuaciones de Lagrange:

```
n = k/(a_j**(3/2))
apunto_s = 2/(n*a_j) * DRlambda_o
epunto_s = -(1-e_j**2)**(1/2)/(n*a_j**2*e_j) * DRvarpi - (1-e_j**2)**(1/2)/
 ↪(n*a_j**2*e_j) * (1-(1-e_j**2)**(1/2) ) * DRlambda_o
Omegapunto_s = 1/(n*a_j**2*(1-e_j**2)**(1/2)*sy.sin(i_j))*DRi
ipunto_s = -sy.tan(i_j/2) / (n*a_j**2*(1-e_j**2)**(1/
 ↪2))*(DRvarpi+DRlambda_o) - 1/(n*a_j**2*(1-e_j**2)**(1/2)*sy.
 ↪sin(i_j))*DROmega
varpipunto_s = (1-e_j**2)**(1/2)/(n*a_j**2*e_j)*DRe + sy.tan(i_j/2)/
 ↪(n*a_j**2*(1-e_j**2)**(1/2))*DRi
lambdaopunto_s = -(1-e_j**2)/(n*a_j**2*e_j)*DRe -  2/(n*a_j)*DRa+ n
lambdappunto_s = k/a_N**(3/2)

apunto_f = sy.
 ↪lambdify([a_j,e_j,i_j,varpi_j,Omega_j,lambda_j,lambda_N],apunto_s)
epunto_f = sy.lambdify([a_j,e_j,i_j,varpi_j,Omega_j,lambda_j,lambda_N],␣
 ↪epunto_s)
varpipunto_f = sy.lambdify([a_j,e_j,i_j,varpi_j,Omega_j,lambda_j,lambda_N],␣
 ↪varpipunto_s)
Omegapunto_f = sy.lambdify([a_j,e_j,i_j,varpi_j,Omega_j,lambda_j,lambda_N],␣
 ↪Omegapunto_s)
ipunto_f = sy.lambdify([a_j,e_j,i_j,varpi_j,Omega_j,lambda_j,lambda_N],␣
 ↪ipunto_s)
lambdajpunto_f = sy.
 ↪lambdify([a_j,e_j,i_j,varpi_j,Omega_j,lambda_j,lambda_N],lambdaopunto_s)
lambdaipunto_f = sy.
 ↪lambdify([a_j,e_j,i_j,varpi_j,Omega_j,lambda_j,lambda_N],lambdappunto_s)

def lagrange_sist(t, variables):
    a,e,i,varpi,Omega,lambda_o,lambda_p = variables
    apunto = apunto_f(a,e,i,varpi,Omega,lambda_o,lambda_p)
    epunto = epunto_f(a,e,i,varpi,Omega,lambda_o,lambda_p)
    ipunto = ipunto_f(a,e,i,varpi,Omega,lambda_o,lambda_p)
    varpipunto = varpipunto_f(a,e,i,varpi,Omega,lambda_o,lambda_p)
    Omegapunto = Omegapunto_f(a,e,i,varpi,Omega,lambda_o,lambda_p)
    lambdajpunto = lambdajpunto_f(a,e,i,varpi,Omega,lambda_o,lambda_p)
    lambdaipunto = lambdaipunto_f(a,e,i,varpi,Omega,lambda_o,lambda_p)
    return␣
 ↪[apunto,epunto,ipunto,varpipunto,Omegapunto,lambdajpunto,lambdaipunto]

y0 = [a_j0,0.1,np.deg2rad(5),np.deg2rad(130),np.deg2rad(200),np.
 ↪deg2rad(30),0]
t = np.arange(0, 365*1e7, 365*1e5)
evolucion = odeint(lagrange_sist, y0, t, tfirst=True)
```

Los resultados son:



Ejemplo 4: resonancia 2:1 con Júpiter

Este ejemplo está basado en el ejemplo 6.9.2 de Murray y Dermott.

El procedimiento es el mismo, solo que tenemos que cargar también los términos resonantes, utilizando `df.list_resonance_terms`.

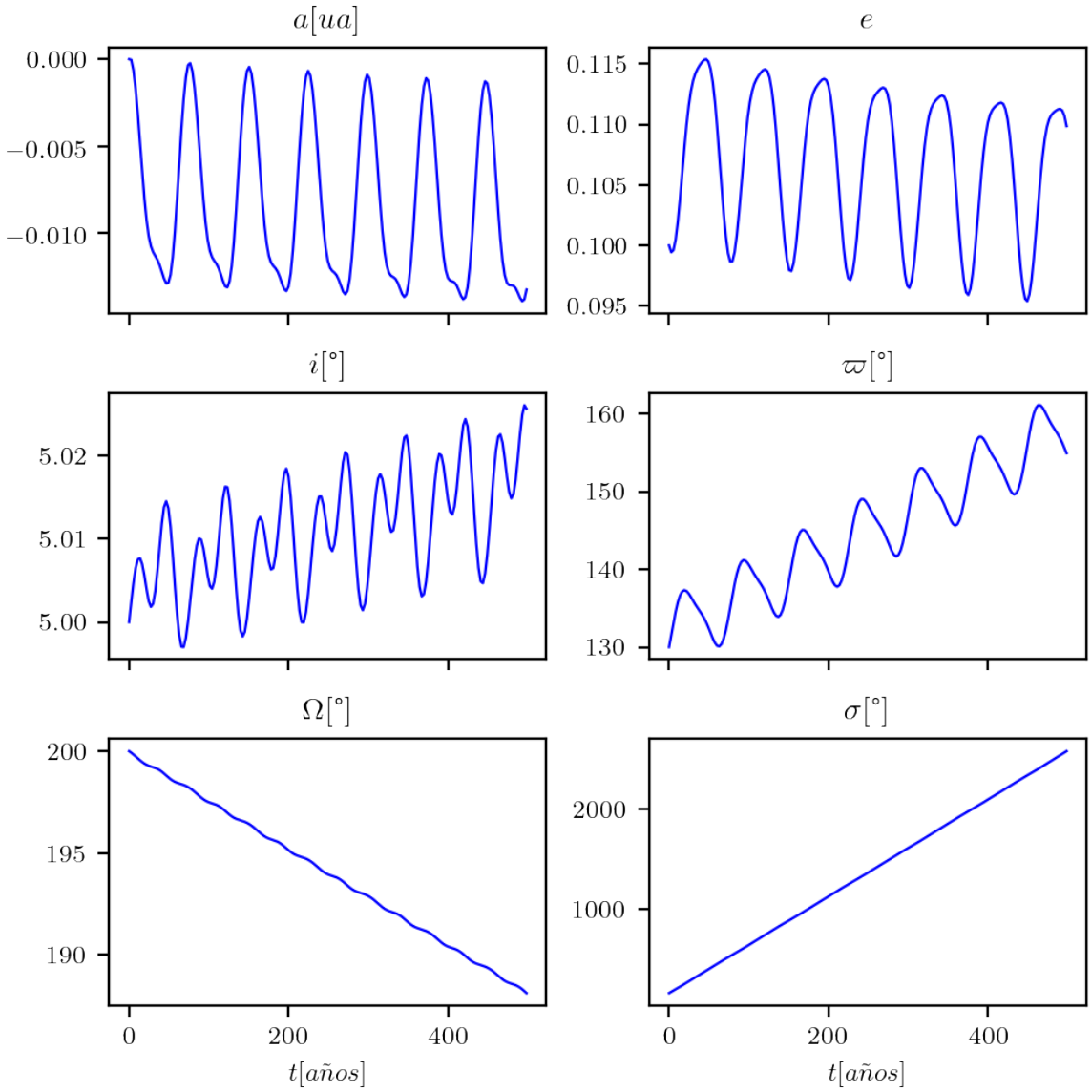
```
a_i,a_j,e_i,e_j,i_i,i_j,varpi_i,varpi_j,Omega_i,Omega_j,lambda_i,lambda_j,
    mu_i,mu_j=sy.symbols('a_i,a_j,e_i,e_j,i_i,i_j,varpi_i,varpi_j,
                    Omega_i,Omega_j,lambda_i,lambda_j,mu_i,mu_j')

#Determinamos las características de la órbita del objeto perturbador
#(que consideramos fija)
mu_j =  k**2 * 1/1047.355
varpi_j = 0
Omega_j = 0
e_j = 0.048
i_j = np.deg2rad(0)
a_j = 5.2038

a_i0 = 0.6*a_j

alpha = a_i0/a_j

elementos=[a_i,e_i,i_i,varpi_i,Omega_i,lambda_i,mu_i,a_j,e_j,i_j,varpi_j,
            Omega_j,lambda_j,mu_j]

#Términos seculares y resonantes.
#Se calculan con df.list_resonance_terms(p,p-q,min_order,max_order)
A_res = df.list_secular_terms(0,2) + df.list_resonance_terms(2,1,0,2)

#Escribimos la función perturbadora
R_res_E = fp.funcion_perturbadora(A_res,*elementos,tipo='e',Alpha=alpha)
```

La función perturbadora resulta, hasta segundo orden:

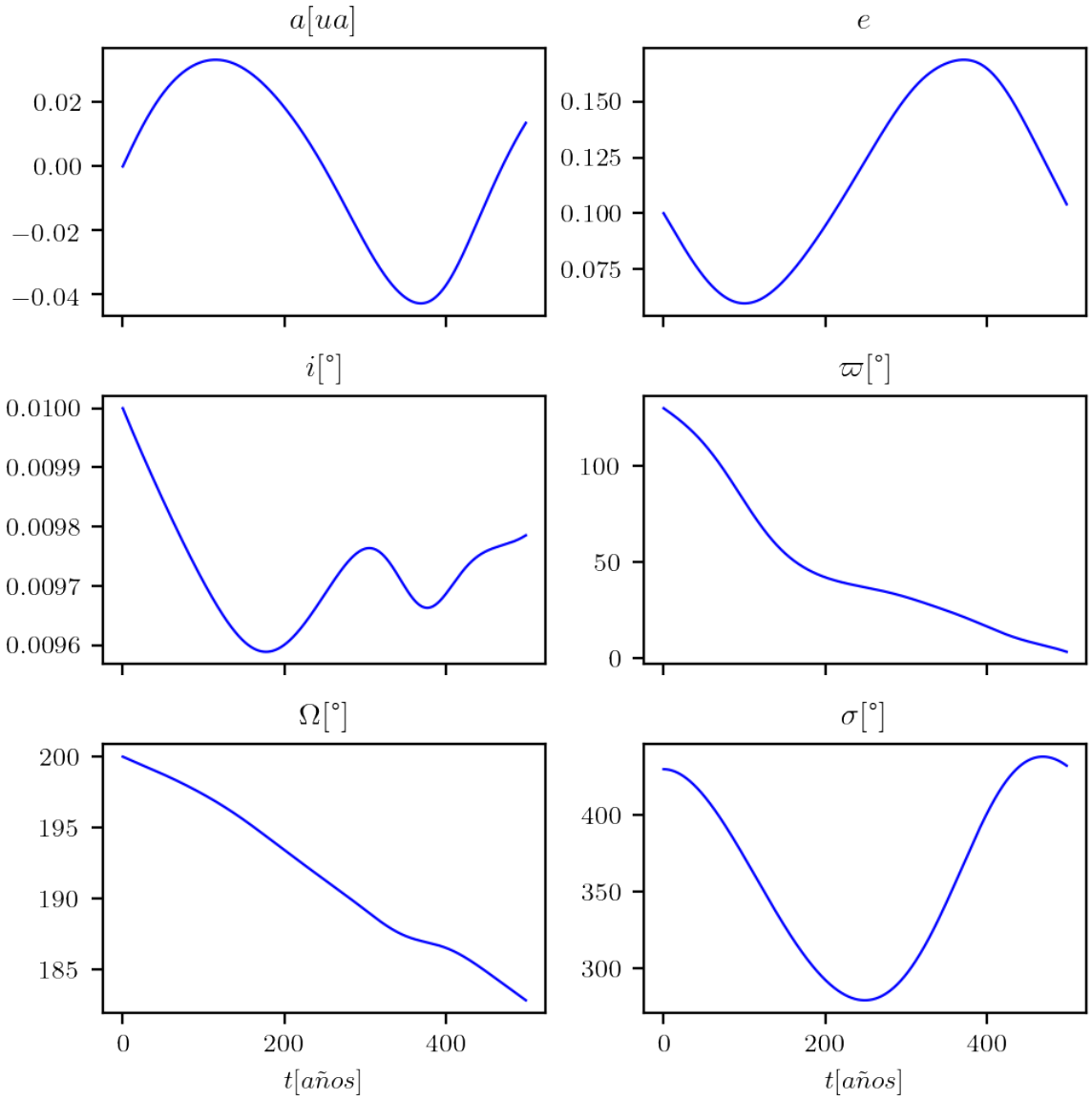
$$\begin{aligned}\mathcal{R}_{res} = {} & 7{,}26203325254126 \cdot 10^{-8} e_i^2 \cos\left(2\lambda_i - 4\lambda_j + 2\varpi_i\right) + 1{,}70482407057367 \cdot 10^{-8} e_i^2 - \\ & - 1{,}16493623654584 \cdot 10^{-9} e_i \cos\left(\varpi_i\right) - 5{,}66456702240496 \cdot 10^{-8} e_i \cos\left(\lambda_i - 2\lambda_j + \varpi_i\right) - \\ & - 1{,}07126272788703 \cdot 10^{-8} e_i \cos\left(2\lambda_i - 4\lambda_j + \varpi_i\right) + \\ & + 3{,}30407361934514 \cdot 10^{-8} \sin^2\left(\frac{i_i}{2}\right) \cos\left(2\Omega_i + 2\lambda_i - 4\lambda_j\right) - \\ & - 6{,}81929628229473 \cdot 10^{-8} \sin^2\left(\frac{i_i}{2}\right) + 9{,}18138223960315 \cdot 10^{-10} \cos\left(\lambda_i - 2\lambda_j\right) + \\ & + 3{,}90318654097469 \cdot 10^{-10} \cos\left(2\lambda_i - 4\lambda_j\right) + 6{,}05529609378104 \cdot 10^{-8}\end{aligned}$$

donde los subíndices i corresponden al asteroide y los j a Júpiter.

Como estamos cerca de una resonancia, puede ser interesante graficar el ángulo crítico, en este caso $\sigma = \lambda_i - 2\lambda_j + \varpi_i$. Resolviendo las ecuaciones, nos queda:



Podemos intentar acercarnos más a la resonancia y se pueden empezar a ver oscilaciones en σ. Por ejemplo, la siguiente figura muestra los resultados para $a_i = 0{,}63a_j$ (más cerca de la resonancia exacta).



Conclusiones

Se generó un módulo de Python `func_pert` a partir del paquete `celmech` que halla y muestra la función perturbadora para un objeto de masa despreciable afectado por perturbaciones internas y externas, siguiendo el algoritmo desarrollado por Ellis y Murray (2000). El código es de uso similar y resultados equivalentes al código `DisturbingFunction.m` para Mathematica. Los resultados se resumen en el notebook adjunto, FuncionPerturbadora.ipynb.

Referencias

[1] Sam Hadden and Daniel Tamayo. celmech: A python package for celestial mechanics. *The Astronomical Journal*, 164(5):179, oct 2022.

[2] Keren M. Ellis and Carl D. Murray. The disturbing function in solar system dynamics. *Icarus*, 147:129–144, 2000.

[3] Carl D. Murray and Stanley F. Dermott. *Solar System Dynamics*. Cambridge University Press, 2000.