UNIVERSITAT DE BARCELONA
Facultat de Matemàtiques i Informàtica

GRAU DE MATEMÀTIQUES

Treball final de grau

CÀLCUL D'ÒRBITES PERIÒDIQUES AMB MÈTODES DE FOURIER

Autora: Núria Jorba i Peña

Director: Dr. Alex Haro Provinciale
Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, 10 de juny de 2024

Abstract

This degree final project develops an algorithm to calculate periodic orbits, both horizontal and vertical, from fixed points in the Restricted Three-Body Problem (RTBP). The research emphasizes the importance of periodic orbits in celestial mechanics and their applications in space missions and astrophysics, aiming to create a robust algorithm for identifying these orbits.

The study is based on classical mechanics, including Lagrangian and Hamiltonian mechanics, the calculus of variations, and the derivation of Lagrange's and Hamilton's equations. This foundation is crucial for understanding the RTBP, where the equations of motion are derived, and the equilibrium solutions and their stability are examined.

The investigation of periodic orbits starts with analyzing linearizations around fixed points and exploring the persistence of families of periodic orbits in Hamiltonian systems. Fourier analysis is introduced, covering Fourier series, the Discrete Fourier Transform (DFT), and the Fast Fourier Transform (FFT) algorithm. These techniques optimize the algorithm's performance. The usual method to investigate these orbits is to use Poincaré sections, which require numerical integration. This project avoids numerical integration with a novel approach.

The project demonstrates the algorithm's reliability in various scenarios, showcasing its potential in solving complex dynamical problems. The source code is provided to ensure reproducibility and transparency, significantly enhancing the understanding and computation of periodic orbits with applications in scientific and engineering fields.

Resum

Aquest treball de final de grau desenvolupa un algorisme per a calcular òrbites periòdiques, tant horitzontals com verticals, a partir de punts fixos en el problema restringit dels tres cossos (RTBP). La investigació subratlla la importància de les òrbites periòdiques en la mecànica celeste i les seves aplicacions en missions espacials i en astrofísica, amb l'objectiu de crear un algorisme robust per identificar aquestes òrbites.

L'estudi es basa en la mecànica clàssica, incloent la mecànica Lagrangiana i Hamiltoniana, el càlcul de variacions i la derivació de les equacions de Lagrange i Hamilton. Aquesta base és crucial per entendre el RTBP, on es deriven les equacions de moviment i s'examinen les solucions d'equilibri i la seva estabilitat.

La investigació de les òrbites periòdiques comença amb l'anàlisi de les linealitzacions al voltant dels punts fixos i l'exploració de la persistència de famílies d'òrbites periòdiques en sistemes Hamiltonians. S'introdueix l'anàlisi de Fourier, que cobreix les sèries de Fourier, la transformada discreta de Fourier (DFT) i l'algorisme de la transformada ràpida de Fourier (FFT). Aquestes tècniques optimitzen el rendiment de l'algorisme. El mètode habitual per investigar aquestes òrbites és utilitzar seccions de Poincaré, que necessiten integració numèrica. Amb aquest mètode l'evitem amb un enfocament nou.

El treball demostra la fiabilitat de l'algorisme en diversos escenaris, mostrant el seu potencial per resoldre problemes dinàmics complexos. Es proporciona el codi font per garantir la reproductibilitat i transparència, millorant significativament la comprensió i el càlcul de les òrbites periòdiques amb aplicacions en camps científics i d'enginyeria.

2020 Mathematics Subject Classification. 3704, 65T50,37J12, 37N05, 70F07.

Agraïments

No puc començar sense expressar la meva profunda gratitud cap al Dr. Alex Haro Provinciale, qui no només m'ha guiat, sinó que també m'ha obert les portes durant aquest darrer any, acceptant amb dedicació tota mena de dubtes i moments d'incertesa. Sense la seva influència, el meu treball no hauria assolit la seva plenitud.

També vull agrair a totes les persones que m'he anat creuant al llarg del grau, tant a les relacions passatgeres com a les que han vingut per quedar-se, de les quals m'enduc un molt bon record i m'emociona poder crear-ne de nous. Als de la feina, que m'han fet costat aquest últim any i m'han ajudat a reafirmar que vull seguir estudiant.

Finalment i especialment, agrair a la meva família i als meus amics de sempre, amb una menció especial per a la Júlia, la Mar i la Laia. Encara que no sempre comprenguin els meus interessos o emocions, em donen suport dia a dia demostrant-me que, passi el que passi i sigui on sigui, em faran costat. Per això, de tot cor, us dedico aquest treball. Us estimo.

De nou, gràcies Alex per ser més que un assessor acadèmic i fer-me acabar amb una molt bona sensació el Grau de Matemàtiques.

Índex

1 Introducció **1**

1.1 Motivacions 1

1.2 Objectius 3

1.3 Estructura de la Memòria 3

2 Introducció a la mecànica clàssica **4**

2.1 Mecànica Lagrangiana 4

2.1.1 Càlcul de variacions. 4

2.1.2 Equacions de Lagrange 8

2.2 Equacions de Hamilton 9

2.3 Sistemes dinàmics continus 10

3 Problema Restringit dels Tres Cossos **12**

3.1 Plantejament del problema 12

3.2 Derivació de les equacions del moviment 14

3.3 Regions on és possible el moviment 16

3.4 Solucions d'equilibri del RTBP. Equilibri del sistema 18

3.5 Estabilitat dels punts d'equilibri 23

4 Òrbites periòdiques **27**

4.1 Anàlisi de les linealitzacions al voltant dels punts fixos 27

4.2 Persistència de les famílies d'òrbites periòdiques en sistemes Hamiltonians . 28

4.3 Cerca de solucions periòdiques a un nivell d'energia fixat 29

4.4 Mètode per a trobar òrbites periòdiques 30

5 Breu introducció a mètodes de Fourier **31**

5.1 Aspectes bàsics de les sèries de Fourier 31

5.2 Transformada de Fourier Discreta. DFT 32

5.3 Algorisme de Cooley-Turkey. FFT 34

5.4 Eliminació del terme de Nyquist i reordenament dels punts 34

6 Algorisme de càlcul d'òrbites periòdiques amb Fourier **36**

6.1 Plantejament del problema 36

6.2 Algorisme resum de resolució 37

6.3 Algorisme implementat detallat 39

6.4 Aplicació 43

6.5 Resultats 46

7 Conclusions **49**

A Teoremes **51**

A.1 Fórmula d'Euler Lagrange a partir del Principi de Hamilton 51

A.2 Teorema de Lyapunov 53

B Codi **57**

1 Introducció

1.1 Motivacions

El problema dels tres cossos

El problema dels tres cossos és un dels problemes més antics de la mecànica clàssica que, avui en dia, segueix fent pensar als matemàtics. És un tema central en la física teòrica des de mitjans del segle XVIII[12]. Al llarg dels anys, s'han obtingut diversos resultats exactes: com la solució del triangle equilàter de Lagrange, i la col·lineal d'Euler[5]. El plantejament del **problema dels tres cossos** és ben senzill:

> *Tres partícules es mouen a l'espai sota la seva atracció gravitatòria mútua. Tenint en compte les seves condicions inicials, determineu el seu moviment posterior.*

Considerem tres partícules de masses m_1, m_2, m_3 que es mouen en un Sistema de Referència Inercial (SRI) en $\mathbb{R}^3$, i que l'única força que actua sobre elles és la seva atracció gravitacional mútua. Denotem per $\mathbf{r_i}$ el vector posició de la partícula i-èsima i $m_i > 0$ la seva massa, per $i \in \{1, 2, 3\}$.

Amb la segona llei de Newton[1] i la llei de gravitació universal[2], s'obté la igualtat:

$$\sum_{i=1}^{3} m_i \ddot{\mathbf{r}}_i = \sum_{j=1, j\neq i}^{3} G\frac{m_i m_j}{r_{ij}^3}(\mathbf{r_j} - \mathbf{r_i}), \qquad \text{on } r_{ij} = ||\mathbf{r_j} - \mathbf{r_i}||. \tag{1.1}$$

Així, per resoldre el problema dels tres cossos, donades unes condicions inicials, per trobar les 3 acceleracions i les 3 posicions, s'ha de resoldre el següent sistema d'equacions diferencials de divuit incògnites ($3masses \cdot (3acc + 3pos) = 3 \cdot 6 = 18$):

$$\begin{cases} m_1\ddot{\mathbf{r}}_1 = \frac{Gm_1m_2}{r_{12}^3}(\mathbf{r_2}-\mathbf{r_1}) + \frac{Gm_3m_1}{r_{13}^3}(\mathbf{r_3}-\mathbf{r_1}) \\ m_2\ddot{\mathbf{r}}_2 = \frac{Gm_2m_1}{r_{12}^3}(\mathbf{r_1}-\mathbf{r_2}) + \frac{Gm_3m_2}{r_{23}^3}(\mathbf{r_3}-\mathbf{r_2}) \\ m_3\ddot{\mathbf{r}}_3 = \frac{Gm_3m_1}{r_{13}^3}(\mathbf{r_1}-\mathbf{r_3}) + \frac{Gm_3m_2}{r_{23}^3}(\mathbf{r_2}-\mathbf{r_3}) \end{cases} \tag{1.2}$$

Utilitzant equacions de conservació i càlcul, es poden reduir el nombre de funcions incògnites del problema dels tres cossos d'ordre divuit fins a ordre quatre, suposant que el moviment

[1] 2na llei de Newton: "*Tot cos sobre el qual actua una força* $\mathbf{F}$ *es mou de tal manera que la variació de la seva quantitat de moviment* $\mathbf{p} = m\mathbf{v}$ *respecte al temps és igual a la força que produeix el moviment.*"

$$\mathbf{F} = \frac{d\mathbf{p}}{dt} = \frac{d(m\mathbf{v})}{dt}$$

[2] Llei de gravetat de Newton o llei de gravitació universal: "*La força d'atracció entre dos cossos, amb masses* m_i *i* m_j *situades en* r_i *i* r_j*, respectivament, és proporcional al producte de les masses* m_i *i* m_j*, i inversament proporcional al quadrat de la distància,* $r_{ij}^2 = ||r_j - r_i||^2$*, que separa els dos cossos*".

$$\mathbf{F}_{ij} = -G\frac{m_i m_j}{r_{ij}^2}\mathbf{u}_{ij},$$

on $\mathbf{u}_{ij} = \frac{r_j - r_i}{||r_j - r_i||}$ és el vector unitari que va del centre de la gravetat d'un cos a l'altre; i $G \approx (6'67430 \pm 0'00015) \cdot 10^{-11} m^3 \cdot s^{-2} \cdot kg^{-1}$ és la constant gravitacional en el Sistema Internacional d'Unitats (SI).

tingui lloc en un pla. Més enllà d'aquestes reduccions, el problema encara és extremadament complicat i manté ocupats als matemàtics des de fa més de dos-cents anys. Es pot consultar aquest estudi en [8].

En aquest treball, s'estudiarà una versió simplificada d'aquest problema: considerant que un dels objectes té massa negligible respecte els altres dos, s'estudiarà l'anomenat problema restringit dels tres cossos (RTBP). El problema és una aproximació important que permet analitzar el moviment d'un cos petit sota la influència gravitacional de dos cossos massius.

A banda de les simplificacions mencionades, el RTBP també té una gran importància tant teòrica com pràctica. Des del punt de vista teòric, aquest problema ajuda a comprendre millor la dinàmica de sistemes de múltiples cossos, proporcionant solucions que són fonamentals per a l'estudi de sistemes més complexos. Des del punt de vista pràctic, les solucions del RTBP són utilitzades en àmbits com la planificació de missions espacials, la predicció de les òrbites de satèl·lits i la comprensió del comportament dinàmic de petits cossos celestes com asteroides i cometes en el sistema solar. Aquest enfocament simplificat permet abordar problemes reals amb un nivell de complexitat manejable, oferint una eina valuosa per a la recerca i l'anàlisi en el camp de la mecànica celeste.

El RTBP ha estat una pedra angular en l'estudi de la dinàmica orbital des de fa dècades. La seva formulació matemàtica no només simplifica els càlculs, sinó que també permet l'obtenció de resultats qualitatius importants sobre l'estabilitat de les òrbites i la predicció del comportament, a llarg termini, dels cossos celestes.

L'estudi detallat d'aquest problema ha conduït a descobriments significatius en la comprensió de les òrbites periòdiques, les ressonàncies orbitals i les regions de caos dinàmic. Això ha permès als investigadors identificar condicions inicials específiques que condueixen a moviments estables o inestables, millorant així la nostra capacitat per predir i gestionar les trajectòries de satèl·lits artificials i sondes espacials.

D'altra banda, el RTBP serveix com a base de models més complexos que involucren múltiples cossos amb masses comparables. Aquests models avançats són essencials per a la investigació en astrodinàmica i per a l'exploració espacial, ja que ofereixen una visió més precisa del comportament de sistemes reals, com ara els sistemes binaris d'estrelles, les llunes dels planetes i els anells planetaris.

Telescopi James Webb

El Telescopi Espacial James Webb (JWST) és una innovadora i imprescindible eina en la investigació astronòmica contemporània. Amb una llançada històrica el 25 de desembre del 2021. A bord del coet *Ariane 5*, el JWST es posiciona com el successor prometedor del Telescopi Espacial Hubble.

La seva ubicació en una òrbita d'halo al voltant del segon punt de Lagrange L_2 del sistema Sol-Terra, a aproximadament 1.500.000 km de distància de la Terra, ofereix una perspectiva única i privilegiada del cosmos. Normalment, un objecte que envolta el Sol i més allunyat que la Terra trigaria més d'un any terrestre a completar la seva òrbita. Però, a prop del punt L_2 l'atracció gravitatòria combinada de la Terra i del Sol permet que una nau espacial orbiti al voltant del Sol el mateix temps que triga la Terra.

La missió del JWST ve motivada per quatre objectius clau que abracen l'essència mateixa de l'exploració astrofísica moderna: buscar la llum de les primeres estrelles i galàxies formades després del Big Bang; estudiar la formació i evolució de les galàxies; millorar la comprensió sobre la formació d'estrelles i planetes; i estudiar els sistemes planetaris i els orígens de la vida.

El telescopi necessita utilitzar un propulsor per tal de mantenir la seva òrbita en L_2 ja que es tracta d'una òrbita inestable, per la qual cosa demana un mantenimient de posició orbital, o el telescopi s'allunyaria d'aquesta configuració.

Aquest treball aproximarà al lector a una primera idea del funcionament d'aquest tipus d'òrbites i com es poden calcular a través de mètodes de Fourier. Alhora, el lector agafarà una primera idea de com funcionen aquests tipus d'òrbites, i com es poden calcular a partir de mètodes de Fourier.

1.2 Objectius

Els objectius del treball són:

- Estudiar el problema restringit dels tres cossos, entenent com funciona la mecànica teòrica per a poder abarcar-lo.
- Explorar la teoria de Fourier.
- Combinar la teoria de Fourier amb la teoria d'òrbites per a obtenir el càlcul d'una família d'òrbites periòdiques.

1.3 Estructura de la Memòria

Aquest treball comença exposant alguns conceptes de mecànica clàssica, veient l'equivalència entre la mecànica lagrangiana i la mecànica hamiltoniana.

En el següent capítol, s'estudia el problema restringit dels tres cossos, on s'aplica aquesta teoria a l'obtenció dels punts fixos d'aquest sistema i a l'estudi de la seva estabilitat.

Es fa un passeig per la teoria que acompanya les òrbites periòdiques i per la teoria de Fourier.

Finalment, s'explica l'algorisme que calcula òrbites periòdiques, desenvolupat i programat en llenguatge de programació C, exposant-ne els resultats.

2 Introducció a la mecànica clàssica

La mecànica estudia el moviment dels cossos i les seves causes. Generalment se suposa que els objectes són puntuals: **partícules** de massa m amb una **posició** definida $\mathbf{q}$.

Els físics moderns diferencien entre: la mecànica "*clàssica*", que abarca un marc ampli; i la mecànica "*newtoniana*", que és un model específic dins d'aquest marc. Per simplificar una mica les coses, hi ha teories com la "*mecànica lagrangiana*"o la "*mecànica hamiltoniana*", que són matemàticament equivalents a la mecànica newtoniana, però presenten conjunts de termes i conceptes diferents. Ambdues són certament clàssiques.

Seguint el llibre *Mathematical Methods of Classical Mechanics*[1], es començarà amb la mecànica lagrangiana i s'acabarà lligant a la hamiltoniana, que és més habitual a l'hora de tractar problemes celestes.

2.1 Mecànica Lagrangiana

La mecànica clàssica de Newton implica que, per un sistema de N partícules sigui necessari resoldre $3N$ EDOs de segon ordre. Aquest problema es complica molt per qualsevol $N \geq 3$. Per això, ens hem de plantejar una descripció alternativa de la mecànica.

La mecànica lagrangiana descriu el moviment en un sistema mecànic en termes de l'espai de configuració[3]. L'espai de configuració d'un sistema mecànic té estructura de varietat diferenciable[4], sobre la qual actua el grup de difeomorfismes. Un sistema mecànic lagrangià ve donat per una varietat i una funció en el seu espai tangent, anomenada lagrangià del sistema. Les solucions són els extremals d'un funcional (acció) definit a partir del lagrangià.

2.1.1 Càlcul de variacions.

El càlcul de variacions s'ocupa dels extrems d'unes funcions el domini de les quals és un espai de dimensions infinites: l'espai de corbes. Aquestes funcions s'anomenen **funcionals**.

En general, un ***funcional*** és qualsevol mapeig des de l'espai de corbes als nombres reals. Un exemple típic de funcional seria la llargada d'una corba en un pla euclidià: si

$$\gamma = \{(t, x) : x(t) = \mathbf{x}, t_0 \leq t \leq t_1\}, \tag{2.1}$$

aleshores $\phi(\gamma) = \int_{t_0}^{t_1} \sqrt{1 + \dot{\mathbf{x}}^2} dt$.

Definició 2.1. *Donada una varietat diferenciable $(\mathbb{R}^n, \tau)$, una **parametrització d'una corba** és una aplicació $\gamma : I \to \mathbb{R}^n$, on $I \subset \mathbb{R}$ és un interval, tal que a cada $t \in I$ li assigna un $\gamma(t) \in \mathbb{R}^n$. Així, es té que $\gamma(I)$ és la **corba parametritzada** per γ.*

[3] **Espai de configuració.** Espai n-dimensional definit per les coordenades generalitzades $\mathbf{q} = (x, y, z)$, que contenen implícitament les lligadures del sistema.

[4] Una **varietat diferenciable** de classe k i dimensió n és un parell $(\mathcal{M}, F)$, on $\mathcal{M}$ és una varietat topològica i F una estructura diferenciable $\mathcal{C}^k$ a $\mathbb{R}^n$.

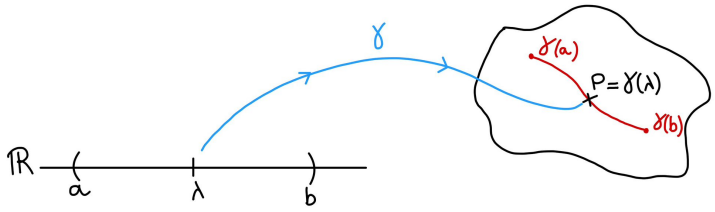


Figura 1: Representació de la parametrització d'una corba.

Sigui $\gamma = \{(t,x) \in \mathbb{R}\times\mathbb{R} : x(t) = \mathbf{x}, t_0 \le t \le t_1\}$, es considera

$$\gamma' = \{(t,x) : x = x(t) + h(t)\} := \gamma + h \tag{2.2}$$

com una *aproximació* de γ; i es defineix

$$\phi(\gamma + h) - \phi(\gamma) \tag{2.3}$$

com l'*increment* de ϕ.

Definició 2.2. *Un* ***funcional*** *es una aplicació ϕ que assigna un número real a cada funció γ en un espai de funcines donat, i.e.*

$$\phi : \{\gamma : [t_0,t_1] \to \mathbb{R}^n, \mathcal{C}^1, \gamma(t_0) = \mathbf{q}_0, \gamma(t_1) = \mathbf{q}_1\} \longrightarrow \mathbb{R}. \tag{2.4}$$

Definició 2.3. *Un funcional ϕ és* ***diferenciable***[5] *si $\phi(\gamma+h)-\phi(\gamma) = F(h)+R(h,\gamma)$, on F depèn linealment de h (i.e. per una γ fixada, $F(h_1+h_2) = F(h_1)+F(h_2)$ i $F(ch) = cF(h)$), i $R(h,\gamma) = O_2(h,\dot{h})$ en el sentit que per $|h| < \epsilon$ i $|\frac{dh}{dt}| < \epsilon$, existeix algun $C \in \mathbb{R}$ tal que tenim $|R| < C\varepsilon^2$. La part lineal de l'increment, $F(h)$ és el* ***diferencial****, de ϕ en γ.*

Teorema 2.4. *Sigui $L : \mathbb{R}\times\mathbb{R}\times\mathbb{R} \to \mathbb{R}$ una funció diferenciable de tres variables. Aleshores, el funcional $\phi(\gamma) = \int_{t_0}^{t_1} L(x,\dot{x},t)dt$ és diferenciable, i la seva derivada ve donada per la fórmula:*

$$D\phi(\gamma)[h] := F(h) = \int_{t_0}^{t_1} \left[\frac{\partial L}{\partial x} - \frac{d}{dt}\frac{\partial L}{\partial \dot{x}}\right] h dt + \left(\frac{\partial L}{\partial \dot{x}} h\right)\Big|_{t_0}^{t_1}, \tag{2.5}$$

on $D\phi(\gamma)[h]$ n'és el ***diferencial****.*

Demostració*.*

$$\begin{aligned}
\phi(\gamma+h) - \phi(\gamma) &= \int_{t_0}^{t_1} [L(x+h, \dot{x}+\dot{h}, t) - L(x,\dot{x},t)]dt \\
&= \int_{t_0}^{t_1} \left[L(x,\dot{x},t) + \frac{\partial L}{\partial x}h + \frac{\partial L}{\partial \dot{x}}\dot{h} + \mathcal{O}_2(h,\dot{h}) - L(x,\dot{x},t)\right]dt \\
&= F(h) + R(h,\gamma),
\end{aligned} \tag{2.6}$$

[5] S'hauria d'especificar la classe de corbes on està definit ϕ i l'espai lineal què conté h. Assumim que els dos espais són de funcions infinitament diferenciables.

on

$$F(h) = \int_{t_0}^{t_1} \Big(\frac{\partial L}{\partial x} h + \frac{\partial L}{\partial \dot{x}} \dot{h}\Big) dt \qquad \text{i} \qquad R(h, \gamma) = \mathcal{O}_2(h, \dot{h}). \tag{2.7}$$

Integrant per parts, es troba que

$$\int_{t_0}^{t_1} \frac{\partial L}{\partial \dot{x}} \dot{h} dt = -\int_{t_0}^{t_1} \frac{d}{dt}\Big(\frac{\partial L}{\partial \dot{x}}\Big) h dt + \Big(h \frac{\partial L}{\partial \dot{x}}\Big)\Big|_{t_0}^{t_1}. \tag{2.8}$$

Així,

$$F(h) = \int_{t_0}^{t_1} \Big[\frac{\partial L}{\partial x} - \frac{d}{dt}\Big(\frac{\partial L}{\partial \dot{x}}\Big)\Big] h dt + \Big(\frac{\partial L}{\partial \dot{x}}\Big) h\Big|_{t_0}^{t_1}. \tag{2.9}$$

□

Definició 2.5. *Un **extremal** d'un funcional diferenciable $\phi(\gamma)$ és una corba γ tal que $D\phi(\gamma)[h] := F(h) = 0$ per a tot h, on h és una deformació del camí de la corba.*

Nota. *Els màxims i mínims del funcional són extremals.*

Lema 2.6. *Sigui $f : [t_0, t_1] \to \mathbb{R}$ una funció contínua en $[t_0, t_1] \subset \mathbb{R}$ que satisfà la igualtat*

$$\int_{t_0}^{t_1} f(t) h(t) dt = 0 \tag{2.10}$$

per tota funció $h : [t_0, t_1] \to \mathbb{R}$ de classe $\mathcal{C}^1(t_0, t_1)$ amb $h(t_0) = h(t_1) = 0$. Aleshores $f(t) \equiv 0$.

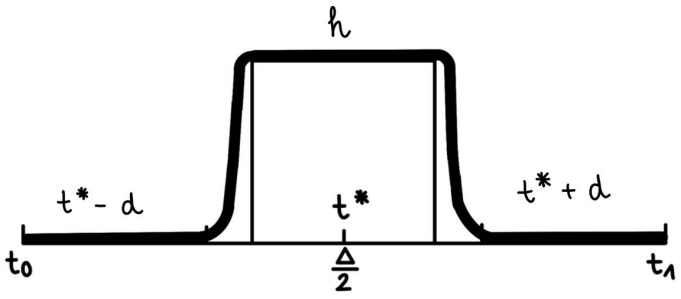


Figura 2: Construcció de la funció h.

Demostració. Es demostra el lema per reducció a l'absurd. Suposem que $f(t^*) > 0$ per algun $t^* \in (t_0, t_1)$. Com que f és contínua, tenim que existeix un $d > 0$ tal que $f(t) > c > 0$ per $t \in \Delta = (t^* - d, t^* + d) \subsetneq (t_0, t_1)$. Sigui $h(t) \in \mathcal{C}^1$ tal que

$$h(t) \begin{cases} = 0 \text{ si} & t \notin \Delta \\ = 1 \text{ si} & t \in \frac{\Delta}{2} = (t^* - \frac{1}{2}d, t^* + \frac{1}{2}d) \\ > 0 \text{ si} & t \in \Delta \end{cases}. \tag{2.11}$$

Aleshores,

$$\int_{t_0}^{t_1} f(t)h(t) \geq d \cdot c > 0, \tag{2.12}$$

que és una contradicció amb la hipòtesi de $\int_{t_0}^{t_1} f(t)h(t) = 0$.

Anàlogament, si se suposa que $f(t^*) < 0$ també s'obté una contradicció. Per tant, es pot concloure que $f(t^*) = 0$ per a tot $t^* \in (t_0, t_1)$, que és equivalent a que f sigui idènticament zero (*i.e.* $f(t) \equiv 0$). □

Teorema 2.7. *La corba γ és un extremal del funcional $\phi(\gamma) = \int_{t_0}^{1} L(x, \dot{x}, t)dt$ a l'espai de corbes passant pels punts $x(t_0) = q_0$ i $x(t_1) = q_1$ si i només si*

$$\frac{d}{dt}\Big(\frac{\partial L}{\partial \dot{x}}\Big) - \frac{\partial L}{\partial x} = 0 \tag{2.13}$$

al llarg de la corba $x(t)$.

Demostració. Es demostra primer la implicació d'esquerra a dreta. Siguin $\gamma : [t_0, t_1] \to \mathbb{R}^n$, amb $\gamma(t_0) = x(t_0) = q_0$ i $\gamma(t_1) = x(t_1) = q_1$; i $h : [t_0, t_1] \to \mathbb{R}^n$ tal que $h(t_0) = h(t_1) = 0$. Del teorema (2.4),

$$F(h) = -\int_{t_0}^{t_1} \left[\frac{d}{dt}\Big(\frac{\partial L}{\partial \dot{x}}\Big) - \Big(\frac{\partial L}{\partial x}\Big)\right] h dt + \Big(\frac{\partial L}{\partial \dot{x}} h\Big)\Big|_{t_0}^{t_1} \tag{2.14}$$

El terme de fora de la integral és igual a zero ja que $h(t_0) = h(t_1) = 0$. Si γ és un extremal, aleshores, per definició, $F(h) = 0$ per tot h tal que $h(t_0) = h(t_1) = 0$. Aleshores, es té que

$$\int_{t_0}^{t_1} f(t)h(t)dt = 0 \ , \tag{2.15}$$

on

$$f(t) = \frac{d}{dt}\Big(\frac{\partial L}{\partial \dot{x}}\Big) - \frac{\partial L}{\partial x}, \tag{2.16}$$

per tot h. Pel lema anterior, $f(t) \equiv 0$.

De dreta a esquerra, si $f(t) \equiv 0$, clarament $F(h) \equiv 0$. □

Definició 2.8 (**Equació d'Euler[6]-Lagrange[7]**). *L'equació*

$$\frac{d}{dt}\Big(\frac{\partial L}{\partial \dot{x}}\Big) - \frac{\partial L}{\partial x} = 0 \tag{2.17}$$

s'anomena ***Equació d'Euler-Lagrange****, pel funcional*

$$\mathcal{L}(x) = \int_{t_0}^{t_1} L(x, \dot{x}, t)dt. \tag{2.18}$$

[6]Leonhard Euler (1707-1783): matemàtic i físic suís.

[7]Joseph Louis Lagrange (1736-1813): matemàtic, físic i astrònom italià.

Ara, sigui $\mathbf{x}$ un vector en l'espai coordenat n-dimensional $\mathbb{R}^n$,

$$\gamma = \{(t, \mathbf{x}) \in \mathbb{R} \times \mathbb{R}^n : \mathbf{x}(t) = \mathbf{x}, t_0 \leq t \leq t_1\} \tag{2.19}$$

una corba en l'epai $(n+1)$-dimensional $\mathbb{R} \times \mathbb{R}^n$, i $L : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$ una funció de $2n+1$ variables. Com abans, es pot ampliar el teorema 2.7 i, demostrant de manera anàloga, s'obté:

Teorema 2.9. *La corba γ és un extremal del funcional $\mathcal{L}(\mathbf{x}) = \int_{t_0}^{t_1} L(\mathbf{x}, \dot{\mathbf{x}}, t)dt$ a l'espai de corbes passant pels punts $\mathbf{x}(t_0) = \mathbf{q}_0$ i $\mathbf{x}(t_1) = \mathbf{q}_1$ si i només si es satisfà l'equació d'Euler-Lagrange al llarg de la corba γ.*

2.1.2 Equacions de Lagrange

Definició 2.10. *L'**energia cinètica** d'un punt de massa m i d'un sistema de n punts de masses són, respectivament:*

$$T = \frac{m\dot{\mathbf{r}}^2}{2}, \qquad T = \sum_{i=1}^{n} \frac{m_i \dot{\mathbf{r}}_i^2}{2} \quad ; \tag{2.20}$$

*on m_i són les masses dels punts i $\dot{\mathbf{r}}_i^2$ les seves velocitats. L'**energia potencial** la definirem com*

$$U(\mathbf{r}) = \sum_{i>j} U_{ij}(|\mathbf{r}_i - \mathbf{r}_j|). \tag{2.21}$$

Es compararà l'equació dinàmica de Newton amb l'equació d'Euler-Lagrange que són, respectivament:

$$\frac{d}{dt}(m_i \dot{r}_i) + \frac{\partial U}{\partial r_i} = 0, \tag{2.22}$$

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\mathbf{x}}} - \frac{\partial L}{\partial \mathbf{x}} = 0. \tag{2.23}$$

Teorema 2.11. *Els moviments del sistema mecànic coincideixen amb els extremals del funcional*

$$\mathcal{L}(\gamma) = \int_{t_0}^{t_1} L dt, \tag{2.24}$$

on $L = T - U$ és la diferència entre l'energia cinètica i la potencial.

Demostració. Com que $U = U(\mathbf{r})$ i $T = T(\dot{\mathbf{r}}) = \sum \frac{m_i \dot{\mathbf{r}}_i^2}{2}$, aleshores:

$$\partial L/\partial \dot{\mathbf{r}}_i = \partial T/\partial \dot{\mathbf{r}}_i = m_i \dot{\mathbf{r}}_i \tag{2.25}$$

$$\partial L/\partial \mathbf{r}_i = -\partial U/\partial \mathbf{r}_i. \tag{2.26}$$

□

Corol·lari 2.12. *Siguin $(q_1, ..., q_{3n})$ coordenades qualssevulla en l'espai de configuració d'un sistema de n masses. Aleshores l'evolució de $\mathbf{q}$ amb el temps ve donada per les equacions d'Euler-Lagrange.*

Demostració. Pel teorema anterior, un moviment és un extremal del funcional $\int L dt$. Per tant, en qualsevol sistema de coordenades, l'equació d'Euler-Lagrange escrita en aquest sistema de coordenades es compleix. □

En mecànica lagrangiana, s'utilitza la següent terminologia:

Definició 2.13. *Siguin T i U les energies cinètiques i potencials del sistema. El* ***lagrangià*** *(o* ***funció de Lagrange****) ve definit per $L(\mathbf{q}, \dot{\mathbf{q}}) = T - U$, on $\mathbf{q}$ i $\dot{\mathbf{q}}$ són, respectivament, les coordenades i velocitats generalitzades; $\partial L/\partial \dot{q}_i = p_i$ el* ***moment generalitzat****; $\partial L/\partial q_i$ les forces generalitzades; $S = \int_{t_0}^{t_1} L(\mathbf{q}, \dot{\mathbf{q}}, t)dt$ és l'****acció****, i $(d(\partial L/\partial \dot{q}_i)/dt) - (\partial L/\partial q_i) = 0$ són les* ***equacions de Lagrange****.*

El teorema 2.11 s'anomena **Principi de Hamilton o principi de mínima acció**. *El moviment del sistema a l'espai de configuració és tal que l'acció S, que és una integral de línia a l'espai de configuració, té un valor estacionari.*

Aquest principi, adopta la següent forma:

$$\delta S = \delta \int_{t_0}^{t_1} L(\mathbf{q}, \dot{\mathbf{q}}, t)dt = 0. \tag{2.27}$$

En l'annex A.1, hi ha una demostració de com extreure les equacions d'Euler Lagrange d'aquest principi.

2.2 Equacions de Hamilton

Considerem el sistema d'equacions de Lagrange $\dot{\mathbf{p}} = \frac{\partial L}{\partial \mathbf{q}}$, on $\mathbf{p} = \frac{\partial L}{\partial \dot{\mathbf{q}}}$, amb una funció lagrangiana donada $L : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$, que assumirem convexa[8] respecte al segon argument $\dot{\mathbf{q}}$. I que, per $\bar{\mathbf{q}}_0, \bar{t}_0$ fixats, l'aplicació $\dot{\mathbf{q}} \longmapsto \mathbf{p} = \frac{\partial L}{\partial \dot{\mathbf{q}}}(\bar{\mathbf{q}}_0, \dot{\mathbf{q}}, \bar{t}_0)$ n'és difeomorfisme[9]; de tal manera que es pot escriure tant $\mathbf{p} = \mathbf{p}(\mathbf{q}, \dot{\mathbf{q}}, t)$ com $\dot{\mathbf{q}} = \dot{\mathbf{q}}(\mathbf{q}, \mathbf{p}, t)$.

Definició 2.14. *Sigui U un subconjunt obert de $\mathbb{R}^{2n}$ i sigui $H \in \mathcal{C}^2(U)$, on $H = H(\mathbf{x}, \mathbf{y})$ amb $\mathbf{x}, \mathbf{y}, \in \mathbb{R}^n$. Un sistema tal que*

$$\begin{cases} \dot{\mathbf{x}} &= \frac{\partial H}{\partial \mathbf{y}} \\ \dot{\mathbf{y}} &= -\frac{\partial H}{\partial \mathbf{x}} \end{cases}, \tag{2.28}$$

amb

$$\frac{\partial H}{\partial \mathbf{x}} = \Big(\frac{\partial H}{\partial x_1}, ..., \frac{\partial H}{\partial x_n}\Big)^T, \qquad \frac{\partial H}{\partial \mathbf{y}} = \Big(\frac{\partial H}{\partial y_1}, ..., \frac{\partial H}{\partial y_n}\Big)^T, \tag{2.29}$$

s'anomena ***sistema Hamiltonià*** *H amb n graus de llibertat en U.*

[8] Una funció real f definida en un interval és **funció convexa** si per dos punts qualssevol x i y en un domini $\mathcal{C}$ i qualsevol t a $[0, 1]$, es compleix: $f(tx + (1-t)y) \leq tf(x) + (1-t)f(y)$.

[9] Un **difeomorfisme**, (o **isomorfisme de varietats diferenciables**) és una bijecció $f : M \to N$ tal que f i f^{-1} són diferenciables.

Teorema 2.15. *El sistema de les equacions de Lagrange és equivalent al sistema d'equacions de primer ordre (equacions de Hamilton)*

$$\begin{cases} \dot{\mathbf{p}} = -\frac{\partial H}{\partial \mathbf{q}} \\ \dot{\mathbf{q}} = \frac{\partial H}{\partial \mathbf{p}} \end{cases}, \tag{2.30}$$

on $H(\mathbf{q}, \mathbf{p}, t) = \mathbf{p}\dot{\mathbf{q}} - L(\mathbf{q}, \dot{\mathbf{q}}, t) = \mathbf{p}\dot{\mathbf{q}}(\mathbf{q}, \mathbf{p}, t) - L(\mathbf{q}, \dot{\mathbf{q}}, t)$ *és la* ***transformació de Legendre*** *de la funció lagrangiana vista com a funció de* $\dot{\mathbf{q}}$.

Demostració. Per definició, la transformada de Legendre de $L(\mathbf{q}, \dot{\mathbf{q}}, t)$ respecte a $\dot{\mathbf{q}}$ és el Hamiltonià $H(\mathbf{q}, \mathbf{p}, t) = \mathbf{p}\dot{\mathbf{q}} - L(\mathbf{q}, \dot{\mathbf{q}}, t)$, on $\dot{\mathbf{q}}$ està expressada en funció de $\mathbf{p}$ per la fórmula $\mathbf{p} = \frac{\partial L}{\partial \dot{\mathbf{q}}}$, que depèn dels paràmetres $\mathbf{q}$ i t.

La derivada total del Hamiltonià

$$dH = \frac{\partial H}{\partial \mathbf{p}} d\mathbf{p} + \frac{\partial H}{\partial \mathbf{q}} d\mathbf{q} + \frac{\partial H}{\partial \mathbf{t}} d\mathbf{t}, \tag{2.31}$$

és igual a la derivada total de $\mathbf{p}\dot{\mathbf{q}} - L$ per $\mathbf{p}(t) = \frac{\partial L}{\partial \dot{\mathbf{q}}}(\mathbf{q}, \dot{\mathbf{q}}, t)$

$$dH = \dot{\mathbf{q}} d\mathbf{p} - \frac{\partial L}{\partial \mathbf{q}} d\mathbf{q} - \frac{\partial L}{\partial \mathbf{t}} d\mathbf{t}. \tag{2.32}$$

Ambdues expressions per dH han de coincidir. Aleshores,

$$\frac{\partial H}{\partial \mathbf{p}} = \dot{\mathbf{q}}, \tag{2.33}$$

$$\frac{\partial H}{\partial \mathbf{q}} = -\frac{\partial L}{\partial \mathbf{q}} = -\dot{\mathbf{p}}, \tag{2.34}$$

$$\frac{\partial H}{\partial \mathbf{t}} = -\frac{\partial L}{\partial \mathbf{t}}. \tag{2.35}$$

Aplicant les equacions de Lagrange $\dot{\mathbf{p}} = \frac{\partial L}{\partial \mathbf{q}}$, s'obtenen les equacions de Hamilton. S'ha vist que, si $\mathbf{q}(t)$ satisfà les equacions de Lagrange, aleshores $(\mathbf{p}(t), \mathbf{q}(t))$ satisfà les equacions de Hamilton. El resultat invers es demostra anàlogament. Per tant, els sistemes de Hamilton i de Lagrange són equivalents. □

Els sistemes Hamiltonians són una classe important de sistemes dinàmics continus, descrits per un sistema d'equacions diferencials.

2.3 Sistemes dinàmics continus

Definició 2.16. *Els* ***sistemes dinàmics continus*** *són aquells on el temps és una variable contínua, es considera* $t \in \mathbb{R}$. *Es defineixen en termes d'un sistema d'EDOs, equacions diferencials ordinàries autònomes:* $\dot{\mathbf{x}} = \mathbf{X}(\mathbf{x})$, *per* $\mathbf{x} = (x_1, ..., x_n) \in \mathbb{R}^n$, *on* $X : \mathcal{U} \subset \mathbb{R}^n \to \mathbb{R}^n$ *és un camp vectorial* $\mathcal{C}^r$*-diferenciable.*

Definició 2.17. *Donat un* $\mathbf{x_0} \in \mathcal{U}$, *el* ***problema de Cauchy*** *(o problema del valor inicial)*

$$\{\dot{\mathbf{x}} = X(\mathbf{x}), \mathbf{x}(\mathbf{0}) = \mathbf{x_0}\} \tag{2.36}$$

consisteix en trobar una funció diferenciable $\phi : I \to \mathbb{R}^n$ *tal que* $\phi(0) = \mathbf{x_0}$ *i per a tot* $t \in I$*,*

$$\phi(t) \in \mathcal{U} \ i \ \frac{d\phi}{dt} = X(\phi(t)). \tag{2.37}$$

S'escriviu la solució del problema com el **flux** $\varphi(t; x_0)$, on φ depèn de $\mathbf{x}_0 \in \mathcal{U}$ i $t \in I(x_0) = \{t \in \mathbb{R} | (t, \mathbf{x}_0) \in \mathbb{R} \times \mathcal{U}\}$, que és l'interval de definició de la solució maximal.

Definició 2.18. *El* ***flux*** *en un conjunt* $\mathcal{U}$ *és una aplicació* $\mathcal{C}^r$*-diferenciable*

$$\varphi : \ \mathbb{R} \times \mathcal{U} \longrightarrow \mathcal{U} \tag{2.38}$$

tal que, per a tot $\mathbf{x} \in \mathcal{U}$ *i per a tots* $s, t \in \mathbb{R}$ *es compleix que* $\varphi(\mathbf{x}, 0) = \mathbf{x}$ *i que*

$$\varphi_s \circ \varphi_t(\mathbf{x}) = \varphi(\varphi(\mathbf{x}, t), s) = \varphi(\mathbf{x}, s + t) = \varphi_{s+t}(\mathbf{x}). \tag{2.39}$$

Definició 2.19. *Donada una condició inicial* x_0*, l'òrbita corresponent és* $\{\phi_t(x_0)\}_{t \in I}$*, on* $I \subset \mathbb{R}$ *és un interval. Un* ***punt fix*** *d'un sistema dinàmic continu és un punt tal que el punt és la seva pròpia òrbita i.e.* $\phi_t(x) = x$ *per tot* $x \in \mathbb{R}$*. Això només pot passar si* $X(\phi_t(x_0)) = X(x) = 0$*. Una òrbita* $\{\phi_t(x)\}_{t \in \mathbb{R}}$ *és* ***periòdica*** *si existeix un* $T > 0$*, anomenat* ***període*** *tal que*

$$\begin{aligned} &\phi_T(x) = x \\ &\phi_t(x) \neq x, \ per \ 0 < t < T. \end{aligned} \tag{2.40}$$

Un conjunt de condicions inicials $A \subset \mathbb{R}^n$ *és un* ***conjunt invariant*** *si*

$$\phi_t(x) \in A \ \ \forall t \in \mathbb{R}, \qquad \forall x \in A. \tag{2.41}$$

Definició 2.20. *Una* ***varietat invariant*** *és un conjunt invariant que és una varietat*[10]*. Hi ha varietats invariants del tipus* ***estables***, ***inestables*** *o* ***centrals***, *associades a un objecte invariant. Donat un conjunt invariant* A*, el seu* ***conjunt estable*** *(resp.* ***inestable****) és el conjunt* $W^s(A)$ *(resp.* $W^u(A)$*) de condicions inicials tendint assimptòticament a l'objecte avançant (resp. retrocedint) en el temps.*

$$W^s(A) = \{x : dist(\phi_t(x), A) \overset{t \to +\infty}{\to} 0\} \tag{2.42}$$

$$W^u(A) = \{x : dist(\phi_t(x), A) \overset{t \to -\infty}{\to} 0\}. \tag{2.43}$$

Observació 2.21. La majoria de casos on A és una varietat, aquests conjunts també ho són i s'anomenen **varietat estable** i **varietat inestable**.

[10] Una **varietat** és un espai topològic en què per a tots els punts existeix un entorn homeomorf a l'espai euclidià.

3 Problema Restringit dels Tres Cossos

Un primer pas elemental per a poder enviar el satèl·lit James Webb a l'espai és fer un anàlisi del problema restringit dels tres cossos.

3.1 Plantejament del problema

El problema restringit dels tres cossos (RTBP), on es considera negligible la massa d'una de les partícules, és un cas particular del problema dels tres cossos (TBP). En aquest cas, el moviment dels dos objectes primaris té lloc en un pla i el problema es planteja com:

> *Siguin dues grans masses, m_1 i m_2, seguint una trajectòria kepleriana al voltant del seu Centre de Masses. S'hi afegeix una tercera massa m_3 tal que $m_3 \ll m_1$ i $m_3 \ll m_2$; s'ha de deduir el moviment d'aquest tercer objecte, que experimenta les forces gravitacionals d'ambdós cossos més grans.*

Tot i la simplicitat del RTBP envers el TBP, el RTBP és un bon model per a alguns sistemes com, per exemple, el cas d'un satèl·lit en el sistema Terra-Lluna, que és el problema en el que se centra aquest treball. L'excentricitat[11] de l'òrbita de la Lluna al voltant de la Terra és propera a zero, fent d'aquesta una òrbita essencialment circular. D'ara en endavant, es prendrà l'òrbita com si fos perfectament circular.

Es vol trobar la trajectòria en l'espai $\mathbf{q}(t)$ de la partícula de massa negligible o, més ben dit, escriure les equacions diferencials del moviment que, resolent-les, permeten trobar aquesta trajectòria. Per això, es consideraran diferents sistemes de coordenades.

Es consideren m_1 i m_2 girant al voltant del seu centre de masses per la força de gravitació (cossos **primaris**) en el Sistema de Referència (SR) $\bar{X}$-$\bar{Y}$; i m_3 la massa menyspreable, movent-se en el mateix pla. Sigui r_{12} la distància entre els cossos primaris, $\mathbf{r}_i$ el vector que va de l'origen a m_i, $i \in \{1, 2\}$, i ω la velocitat angular a la qual es mouen m_1 i m_2.

Definició 3.1. *La **velocitat angular** és una mesura de la velocitat de rotació. Es defineix com l'angle girat per una unitat de temps i es designa mitjançant ω. La seva unitat del SI són (rad/s).*

Les dues masses orbiten al mateix ritme al voltant del mateix Centre de Masses (a velocitat angular constant), es pot apreciar en la Figura 3. Així, r_{12} es manté constant i es poden prendre unitats unitàries:

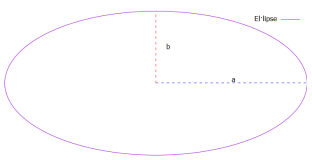
$$r_{12} = r_1 + r_2 = m_1 + m_2 = 1. \tag{3.1}$$

Posant $\mu = \frac{m_2}{m_1+m_2} = m_2$ i tenint en compte (3.1), s'obté

$$r_1 = \frac{r_{12}m_2}{m_1 + m_2} = r_{12}\mu = \mu; \qquad r_2 = 1 - r_1 = 1 - \mu. \tag{3.2}$$

Així, $\frac{r_1}{r_2} = \frac{\mu}{1-\mu} = \frac{m_2}{m_1}$.

[11] En qualsevol el·lipse, on la longitud del semieix major és a i el semieix menor és b, l'**excentricitat** ve donada per $e = \sqrt{1 - \frac{a^2}{b^2}} \in [0, 1)$. La de l'òrbita de la Lluna oscil·la entre 0.026 i 0.077.



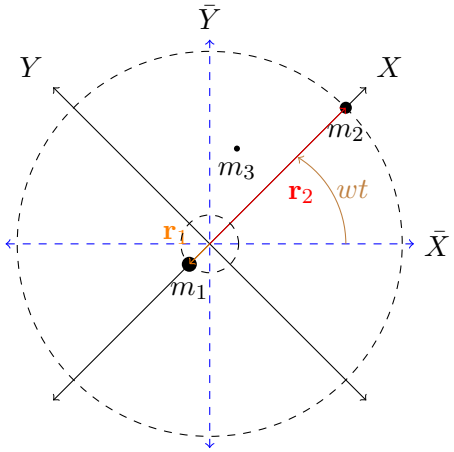


Figura 3: Representació del problema restringit dels tres cossos.

De la tercera llei de Kepler, $M(r_1 + r_2)\omega^2 = G\frac{mM}{(r_1+r_2)^2}$, s'obté $\omega^2 = \frac{G(m_1+m_2)}{r_{12}^3}$. Així, gràcies a (3.1), es dedueix que també es pot agafar $G = \omega^2 = ||\omega||^2 = 1$.

Sense pèrdua de generalitat, s'assumeix $m_1 > m_2$ i, per tant, prenent $m = m_1 + m_2 = m_1 + \mu = 1$, s'obté $\mu \leq 0.5$. Es pot prendre un Sistema de Referència Inercial[12] (SRI) amb eixos $\bar{X}$-$\bar{Y}$; o, com que ω és constant, un SR Rotatòri (SRR) amb eixos X-Y. El lligam entre els dos SR ve donat per la següent matriu de rotació:

$$A_t = \begin{pmatrix} \cos(\omega t) & -\sin(\omega t) & 0 \\ \sin(\omega t) & \cos(\omega t) & 0 \\ 0 & 0 & 1 \end{pmatrix}; \qquad \bar{\mathbf{q}} = \begin{pmatrix} \bar{x} \\ \bar{y} \\ \bar{z} \end{pmatrix} = A_t \begin{pmatrix} x \\ y \\ z \end{pmatrix} = A_t \mathbf{q}. \tag{3.3}$$

En el sistema rotatori o sinòdic X-Y, les masses m_1 i m_2 estan situades, respectivament, a:

$$\begin{cases} \mathbf{q}^1 = (x_1, y_1, z_1)^T = (-\mu, 0, 0)^T \\ \mathbf{q}^2 = (x_2, y_2, z_2)^T = (1-\mu, 0, 0)^T \end{cases}. \tag{3.4}$$

En el sistema inercial o sideri $\bar{X}$-$\bar{Y}$, les masses m_1 i m_2 estan situades, respectivament, a:

$$\begin{cases} \bar{\mathbf{q}}^1 = (\bar{x}_1, \bar{y}_1, \bar{z}_1)^T = A_t(x_1, y_1, z_1)^T = A_t(-\mu, 0, 0)^T = (-\mu\cos(t), -\mu\sin(t), 0)^T \\ \bar{\mathbf{q}}^2 = (\bar{x}_2, \bar{y}_2, \bar{z}_2)^T = A_t(1-\mu, 0, 0)^T = (\ (1-\mu)\cos(t),\ (1-\mu)\sin(t), 0)^T \end{cases}.$$

Notant que:

$$\dot{A}_t = \omega \begin{pmatrix} -\sin(\omega t) & -\cos(\omega t) & 0 \\ \cos(\omega t) & -\sin(\omega t) & 0 \\ 0 & 0 & 0 \end{pmatrix} = wA_tN, \text{ on } N = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \tag{3.5}$$

[12] Un **Sistema de Referència Inercial** (SRI) o **galileà**, també anomenat **sideri**, és un sistema de referència en el qual les relacions espacials (determinades per escales rígides en repòs en el sistema) són euclidianes i, a més a més, existeix un temps universal en termes del qual les partícules lliures estan en repòs o segueixen línies rectes a velocitat constant (és a dir, les partícules lliures satisfan la primera llei de Newton).

es calcula la derivada de $\bar{q}$:

$$\dot{\bar{\mathbf{q}}} = \begin{pmatrix} \dot{\bar{x}} \\ \dot{\bar{y}} \\ \dot{\bar{z}} \end{pmatrix} = \dot{A}_t \begin{pmatrix} x \\ y \\ z \end{pmatrix} + A_t \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} \overset{(3.5)}{=} A_t N w \begin{pmatrix} x \\ y \\ z \end{pmatrix} + A_t \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} \tag{3.6}$$
$$\overset{\omega=1}{=} A_t \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + A_t \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} = A_t \begin{pmatrix} -y \\ x \\ 0 \end{pmatrix} + A_t \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} = A_t \begin{pmatrix} -y + \dot{x} \\ x + \dot{y} \\ \dot{z} \end{pmatrix}.$$

3.2 Derivació de les equacions del moviment

Per trobar les equacions del moviment, es fan servir les equacions d'Euler-Lagrange (2.17), provinents del principi de Hamilton, esmentades en el primer capítol:

$$\frac{\partial L}{dq^i} - \frac{d}{dt}\left(\frac{\partial L}{d\dot{q}^i}\right) = 0, \tag{3.7}$$

on $L = T - U$, sent T l'energia cinètica del sistema i U l'energia potencial.

En el Sistema de Referència Inercial, el lagrangià ve donat per:

$$\mathcal{L}(\bar{\mathbf{q}}, \dot{\bar{\mathbf{q}}}, t) = \mathcal{T}(\bar{\mathbf{q}}, \dot{\bar{\mathbf{q}}}) - \mathcal{U}(\bar{\mathbf{q}}, t), \tag{3.8}$$

on

$$\mathcal{T}(\bar{\mathbf{q}}, \dot{\bar{\mathbf{q}}}) = \frac{1}{2}(\dot{\bar{x}}^2 + \dot{\bar{y}}^2 + \dot{\bar{z}}^2); \qquad \mathcal{U}(\bar{\mathbf{q}}, t) = -\frac{Gmm_1}{\bar{r}_1(t)} - \frac{Gmm_2}{\bar{r}_2(t)}. \tag{3.9}$$

Tenint en compte que les matrius de rotació són ortogonals ($A_t^T A_t = A_t A_t^T = Id$), l'energia cinètica del sistema és:

$$\begin{aligned} \mathcal{T}(\bar{\mathbf{q}}, \dot{\bar{\mathbf{q}}}) &= \frac{1}{2} m \mathbf{v}^2 \overset{m=1}{=} \frac{1}{2}\dot{\bar{\mathbf{q}}}^2 = \frac{1}{2}(\dot{\bar{x}}^2 + \dot{\bar{y}}^2 + \dot{\bar{z}}^2) = \frac{1}{2}\dot{\bar{\mathbf{q}}}^T \dot{\bar{\mathbf{q}}} \\ &= \frac{1}{2}\begin{pmatrix} -y + \dot{x}, & x + \dot{y}, & \dot{z} \end{pmatrix} A_t^T A_t \begin{pmatrix} -y + \dot{x} \\ x + \dot{y} \\ \dot{z} \end{pmatrix} = \frac{1}{2}[(-y + \dot{x})^2 + (x + \dot{y})^2 + (\dot{z})^2] \\ &=: T(\mathbf{q}, \mathbf{p}). \end{aligned} \tag{3.10}$$

I, com que

$$\bar{r}_1(t) = ||\bar{\mathbf{q}} - \bar{\mathbf{q}}_1||_2 = ||A_t\mathbf{q} - A_t\mathbf{q}_1||_2 = ||\mathbf{q} - \mathbf{q}_1||_2 = r_1, \tag{3.11}$$
$$\bar{r}_2(t) = ||\bar{\mathbf{q}} - \bar{\mathbf{q}}_2||_2 = ||A_t\mathbf{q} - A_t\mathbf{q}_2||_2 = ||\mathbf{q} - \mathbf{q}_2||_2 = r_2, \tag{3.12}$$

les distàncies es conserven en ambdós SR. Així doncs, el potencial coincideix en els dos sistemes de referència, sent:

$$\mathcal{U}(\bar{\mathbf{q}}, t) = U(\mathbf{q}) = -\frac{Gmm_1}{r_1} - \frac{Gmm_2}{r_2} = -\frac{1-\mu}{r_1} - \frac{\mu}{r_2}, \tag{3.13}$$

on, en la darrera igualtat, s'ha tingut en compte que $G = m = 1$, i on les distàncies són:

$$\begin{cases} r_1 = \sqrt{(x+\mu)^2 + y^2 + z^2} \\ r_2 = \sqrt{(x-(1-\mu))^2 + y^2 + z^2} \end{cases}. \tag{3.14}$$

Aleshores, el lagrangià del SRR és:

$$L = L(\mathbf{q}, \dot{\mathbf{q}}, t) = T(\mathbf{q}, \mathbf{p}) - U(\mathbf{q}) = \frac{1}{2}[(-y + \dot{x})^2 + (x + \dot{y})^2 + (\dot{z})^2] - U(\mathbf{q}). \tag{3.15}$$

Fent servir les equacions d'Euler-Lagrange (2.17), s'obtenen les equacions del moviment lagrangianes en cadascun dels eixos:

$\hookrightarrow i = x \equiv 1$:

$$\frac{1}{2}(2(\dot{y}+x)) - \partial_x U - \frac{d}{d\tau}(\frac{1}{2}2(\dot{x}-y)) = 0 \Rightarrow \frac{d}{d\tau}(\dot{x}-y) = \dot{y}+x-\partial_x U \Rightarrow \ddot{x}-\dot{y} = \dot{y}+x-\partial_x U.$$

$\hookrightarrow i = y \equiv 2$

$$(-y + \dot{x})(-1) - \partial_y U = \frac{d}{d\tau}(x + \dot{y}) \Rightarrow \ddot{y} + \dot{x} = -\dot{x} + y - \partial_y U.$$

$\hookrightarrow i = z \equiv 3$

$$0 - \partial_z U = \frac{d}{d\tau}\dot{z} = \ddot{z}.$$

Així, les equacions del moviment lagrangianes són:

$$\begin{cases} \ddot{x} - 2\dot{y} = x - \partial_x U = -\partial_x \bar{U} \\ \ddot{y} + 2\dot{x} = y - \partial_y U = -\partial_y \bar{U} \\ \ddot{z} = -\partial_z U = -\partial_z \bar{U} \end{cases}, \tag{3.16}$$

on el potencial efectiu ve donat per

$$\bar{U}(x, y, z) = -\frac{1}{2}(x^2 + y^2) + U(x, y, z) = -\frac{1}{2}(x^2 + y^2) - \frac{1-\mu}{r_1} - \frac{\mu}{r_2}. \tag{3.17}$$

Es vol treballar amb la mecànica Hamiltoniana (canviar les velocitats generalitzades pels moments generalitzats corresponents -els moments conjugats-). En en capítol anterior, s'ha vist que la mecànica lagrangiana i la hamiltoniana són equivalents. Per tant, fent servir la Transformada de Legendre s'obtenen les equacions del moviment hamiltonianes:

$$p_j = \frac{\partial L}{\partial \dot{q}^j}. \qquad H(\mathbf{q}, \mathbf{p}, t) = \sum_{j=1}^{n} p_j \dot{q}^j - L(\mathbf{q}, \dot{\mathbf{q}}, t) = \mathbf{p}^T \dot{\mathbf{q}} - L(\mathbf{q}, \dot{\mathbf{q}}, t). \tag{3.18}$$

De

$$\mathbf{q} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}; \quad \mathbf{p} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} = \begin{pmatrix} \frac{\partial L}{\partial \dot{x}} \\ \frac{\partial L}{\partial \dot{y}} \\ \frac{\partial L}{\partial \dot{z}} \end{pmatrix} = \begin{pmatrix} -y + \dot{x} \\ x + \dot{y} \\ \dot{z} \end{pmatrix}, \tag{3.19}$$

s'obté

$$\dot{\mathbf{q}} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} = \begin{pmatrix} p_x + y \\ p_y - x \\ p_z \end{pmatrix}. \tag{3.20}$$

I, amb tot això es troba, finalment, el Hamiltonià:

$$\begin{aligned}
H(\mathbf{q},\mathbf{p},t) &= \mathbf{p}^T\dot{\mathbf{q}} - L(\mathbf{q},\dot{\mathbf{q}},t) = \begin{pmatrix} p_x, & p_y, & p_z \end{pmatrix}\begin{pmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{pmatrix} - L(\mathbf{q},\dot{\mathbf{q}},t)\\
&= \dot{x}p_x + \dot{y}p_y + \dot{z}p_z - \frac{1}{2}(p_x^2+p_y^2+p_z^2) + U(\mathbf{q})\\
&= (p_x+y)p_x + (p_y-x)p_y + p_z^2 - \frac{1}{2}(p_x^2+p_y^2+p_z^2) + U(\mathbf{q})\\
&= \frac{1}{2}p_x^2 + yp_x + \frac{1}{2}p_y^2 - xp_y + \frac{1}{2}p_z^2 + U(\mathbf{q}) + \frac{x^2+y^2}{2} - \frac{x^2+y^2}{2}\\
&= \frac{1}{2}(p_x^2+2yp_x+y^2) + \frac{1}{2}(p_y^2-2xp_y+x^2) + \frac{1}{2}p_z^2 + U(\mathbf{q}) - \frac{x^2+y^2}{2}\\
&= \frac{1}{2}((p_x+y)^2 + (p_y-x)^2 + p_z^2) + \bar{U}(\mathbf{q}) = \frac{1}{2}(\dot{x}^2+\dot{y}^2+\dot{z}^2) + \bar{U}(\mathbf{q}).
\end{aligned}$$

Aleshores, del Hamiltonià

$$H(\mathbf{q},\mathbf{p}) = \frac{1}{2}[(p_x+y)^2 + (p_y-x)^2 + p_z^2] + \bar{U}(\mathbf{q}), \tag{3.21}$$

s'obtenen les equacions del moviment de la mecànica Hamiltoniana (equacions canòniques de Hamilton):

$$\dot{q}^i = \frac{\partial H}{\partial p_i}; \qquad \dot{p}_i = -\frac{\partial H}{\partial q^i}. \tag{3.22}$$

Aleshores, les equacions del moviment de la mecànica hamiltoniana, en aquest cas, són:

$$\begin{cases} \dot{x} &= p_x + y\\ \dot{p_x} &= p_y - x - \partial_x\bar{U}\\ \dot{y} &= p_y - x\\ \dot{p_y} &= -p_x - y - \partial_y\bar{U}\end{cases}, \qquad \begin{cases}\dot{z} &= p_z\\ \dot{p_z} &= -\partial_z\bar{U}\end{cases} \tag{3.23}$$

Nota. Tant amb la forma lagrangiana de les equacions com amb la hamiltoniana, s'obté un sistema independent del temps. Mirat com un sistema dinàmic, és un sistema en un *espai de fase* 6-dimensional vist tant per l'espai de posicions i velocitats, $(\mathbf{q},\dot{\mathbf{q}}) = (x,y,z,\dot{x},\dot{y},\dot{z})$, com pel de posicions i moments, (x,p_x,y,p_y,z,p_z), que són subconjunts de $\mathbb{R}^6$ que exclouen les singularitats en m_1 i m_2.

3.3 Regions on és possible el moviment

Com que les equacions del moviment del RTBP són hamiltonianes i independents del temps, tenen una **integral d'energia** del moviment. Al mirar l'energia en funció de la posició i del moment es fa servir $H = H(\mathbf{q},\mathbf{p})$.

Físicament, la mesura de la posició de les partícules i la seva velocitat determina el valor de l'energia associada amb el moviment de la partícula.

Definició 3.2. *La mecànica celeste fa servir $C = -2H$, anomenada **integral de Jacobi**, i ve donada per*

$$C(\mathbf{q},\dot{\mathbf{q}}) = -(\dot{x}^2+\dot{y}^2+\dot{z}^2) - 2\bar{U}. \tag{3.24}$$

Nota. *Com que H és constant, trivialment C també ho és.*

Definició 3.3. *Definim $\mathcal{M}$ com a* ***varietat d'energia*** *o* ***superfície d'energia*** *un cop fixada la integral d'energia igualant-la a una constant i.e.*

$$\mathcal{M}_{H_0} = \{(\mathbf{q}, \mathbf{p}) \in \mathbb{R}^3 \times \mathbb{R}^3 | H(\mathbf{q}, \mathbf{p}) = H_0\}, \tag{3.25}$$

on H_0 és una constant. Per una H_0 fixada, podem considerar la varietat com una superfície 5-dimensional dins de l'espai de fase 6-dimensional. És una varietat diferenciable si $DH(\mathbf{q}, \mathbf{p})$ no s'anul·la sobre $\mathcal{M}_{H_0}$.

Definició 3.4. *Fixat un nivell d'energia H_0, definim la regió de l'espai de configuració on té sentit el moviment, la* ***regió de Hill***[13]*, com:*

$$R_{H_0} = \{\mathbf{q} = (x, y, z) \in \mathbb{R}^3 | \bar{U}(\mathbf{q}) \leq H_0\}. \tag{3.26}$$

La superfície $Z_{H_0} = \{\mathbf{q} = (x, y, z) \in \mathbb{R}^3 | \bar{U}(\mathbf{q}) = H_0\}$ és la frontera de la regió de Hill, es coneix com a **corba de velocitat 0** i té un paper molt important a l'hora de posar límits al moviment de la partícula. Més enllà de Z_{H_0}, fora de R_{H_0}, el moviment no és possible i la regió es coneix com a *regió prohibida*.

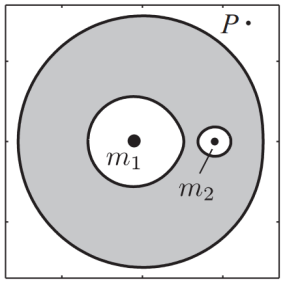
Fent càlcul multivariable (analitzat en les properes seccions) es troba que els punts crítics del Hamiltonià $H(\mathbf{q}, \mathbf{p})$ corresponen als punts fixos del sistema (3.23) i en són cinc: tres punts de sella al llarg de l'eix X i dos punts simètrics fora d'aquest eix. Aquests punts són els punts d'equilibri per a una partícula en el SRR *i.e.* si es col·loqués una partícula en repòs en un d'aquests punts es mantindria al mateix lloc per sempre (no tindria acceleració). En el SRI, aquests punts corresponen a òrbites circulars al voltant de l'origen de coordenades.

Aquests punts crítics s'etiqueten com $L_i, i \in \{1, 2, 3, 4, 5\}$. Ordenant-los de manera que, sigui H_i l'energia de la partícula en repòs en el valor crític L_i, aleshores $H_5 = H_4 > H_3 > H_2 > H_1$; sent L_1 la ubicació del punt d'equilibri amb l'energia més baixa, i L_5 el punt d'equilibri amb més energia.

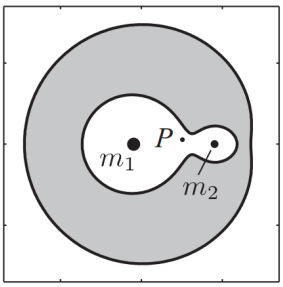
La topologia de les regions de Hill canvia quan H_0 passa pels valors crítics de H. Els punts crítics de H corresponen a punts fixos del camp Hamiltonià. Així doncs, com es veurà en la propera secció, per una μ donada, hi ha 5 configuracions per la regió de Hill, representades en la Figura 4. Aquestes configuracions s'anomenaran ***casos d'energia***, i són:

- **Cas 1**. $H < H_1$, la partícula no es pot moure entre les regions que envolten m_1 i m_2 (no poden moure's d'una regió a l'altra).
- **Cas 2**. $H_1 < H < H_2$, existeix un petit "coll" entre les regions que envolten m_1 i m_2, on hi trobem L_1, permetent a la partícula anar d'una regió a l'altra.
- **Cas 3**. $H_2 < H < H_3$, la partícula es pot moure entre el veïnatge de m_1 i m_2 i l'exterior a través d'un coll, on hi trobem L_2.
- **Cas 4**. $H_3 < H < H_4 = H_5 = -\frac{3}{2}$, la partícula pot passar directament des de la proximitat de m_1 a la regió exterior pel coll on es troba L_3.
- **Cas 5**. $H_4 = H_5 = -\frac{3}{2} < H$, les regions prohibides desapareixen i la partícula es pot moure per tot el pla X-Y.

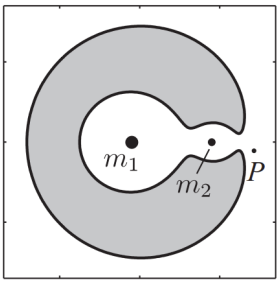
[13] George William Hill (1838-1914): astrònom i matemàtic nord-americà.
Si $H(\mathbf{q}, \mathbf{p}, t) = \frac{1}{2}[(p_x + y)^2 + (p_y - x)^2 + p_z^2] + \bar{U}(\mathbf{q}) = H_0$, al ser el primer sumand ≥ 0, $\bar{U}(\mathbf{q}) \leq H_0$.



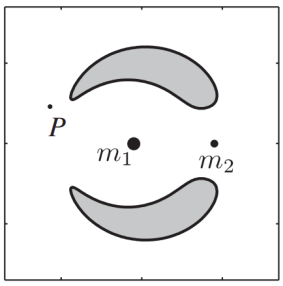
Cas 1. $H < H_1$



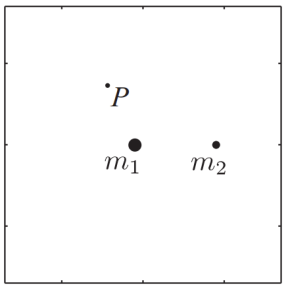
Cas 2. $H_1 < H < H_2$



Cas 3. $H_2 < H < H_3$



Cas 4. $H_3 < H < H_4$



Cas 5. $H_4 < H$.

Figura 4: **Regions on està permès el moviment**. Les regions blanques són les de Hill, on es permet el moviment; i en les regions pintades el moviment està prohibit.

3.4 Solucions d'equilibri del RTBP. Equilibri del sistema

Al final d'aquesta secció els 5 punts crítics de H estaran localitzats, que corresponen a punts fixos del camp Hamiltonià. Per calcular-los, considerem el sistema obtingut en (3.23):

$$\mathbf{x} = \begin{pmatrix} x \\ p_x \\ y \\ p_y \\ z \\ p_z \end{pmatrix} \Rightarrow \dot{\mathbf{x}} = X(\mathbf{x}) = \begin{pmatrix} \dot{x} \\ \dot{p}_x \\ \dot{y} \\ \dot{p}_y \\ \dot{z} \\ \dot{p}_z \end{pmatrix} = \begin{pmatrix} p_x + y \\ p_y - x - \partial_x \bar{U} \\ p_y - x \\ -p_x - y - \partial_y \bar{U} \\ p_z \\ -\partial_z \bar{U} \end{pmatrix}. \tag{3.27}$$

L'equilibri del sistema esdevé en els punts fixos *i.e.* $\dot{\mathbf{x}} = X(\mathbf{x}) = \mathbf{0}$. Així doncs, els punts crítics $(x, y, z) \in \mathbb{R}^3$ de $H(\mathbf{q}, \mathbf{p})$ coincideixen amb els punts crítics de $\bar{U}$:

$$\begin{cases} p_x + y = 0 \\ p_y - x = 0 \\ p_z = 0 \\ p_y - x - \partial_x \bar{U} = 0 \\ -p_x - y - \partial_y \bar{U} = 0 \\ -\partial_z \bar{U} = 0 \end{cases} \Rightarrow \begin{cases} \partial_x \bar{U} = 0 \\ \partial_y \bar{U} = 0 \\ \partial_z \bar{U} = 0 \end{cases} \Leftrightarrow \nabla \bar{U} = 0. \tag{3.28}$$

i.e. els punts (x, p_x, y, p_y, z, p_z) tal que $\nabla \bar{U} = \mathbf{0}$, compleixen $p_x = -y$, $p_y = x$, $p_z = 0$, sent de la forma $(x, -y, y, x, z, 0)$.

Per trobar els punts crítics del Hamiltonià, primer es necessiten les derivades a primer ordre de

$$\bar{U}(x,y,z) = -\tfrac{1}{2}(x^2+y^2) - \tfrac{1-\mu}{r_1} - \tfrac{\mu}{r_2}, \qquad \begin{cases} r_1 = \sqrt{(x+\mu)^2+y^2+z^2} \\ r_2 = \sqrt{(x-(1-\mu))^2+y^2+z^2} \end{cases}.$$

$$\begin{cases} \partial_x \bar{U} = -x + \frac{(1-\mu)(x+\mu)}{r_1^3} + \frac{\mu(x-(1-\mu))}{r_2^3} \\ \partial_y \bar{U} = -y + (\frac{1-\mu}{r_1^3} + \frac{\mu}{r_2^3})y \\ \partial_z \bar{U} = (\frac{1-\mu}{r_1^3} + \frac{\mu}{r_2^3})z \end{cases}. \tag{3.29}$$

Tenint en compte que $r_1, r_2 > 0$ i, al ser $\mu \in (0, \frac{1}{2})$, $\mu > 0$ i $\mu < \frac{1}{2} < 1$, es poden limitar solucions en la segona i la tercera equació, ja es poden trobar els punts on $\nabla \bar{U} = \mathbf{0}$:

$$\begin{cases} \partial_x \bar{U} = -x + \frac{(1-\mu)(x+\mu)}{r_1^3} + \frac{\mu(x-(1-\mu))}{r_2^3} = 0 \\ \partial_y \bar{U} = -y + (\frac{1-\mu}{r_1^3} + \frac{\mu}{r_2^3})y = 0 \\ \partial_z \bar{U} = (\frac{1-\mu}{r_1^3} + \frac{\mu}{r_2^3})z = 0 \end{cases}. \tag{3.30}$$

Així, els punts on $\nabla \bar{U} = \mathbf{0}$ són

$$\begin{cases} \partial_x \bar{U} = -x + \frac{(1-\mu)(x+\mu)}{r_1^3} + \frac{\mu(x-(1-\mu))}{r_2^3} = 0 \\ y = 0 \text{ o bé } -1 + \frac{1-\mu}{r_1^3} + \frac{\mu}{r_2^3} = 0 \\ z = 0 \end{cases}. \tag{3.31}$$

Per tant, els punts fixos del camp Hamiltonià es troben tots en el pla $z = 0$. Es distingeixen entre dos casos, $y = 0$ i $y \neq 0$:

- **Punts col·lineals.** En seran tres, es considera

  $$\boxed{y = 0}, \text{ d'on } \begin{cases} r_1 = \sqrt{(x+\mu)^2} = |x+\mu| \\ r_2 = \sqrt{(x-(1-\mu))^2} = |x-1+\mu| \end{cases}.$$

  Així, definint la funció $f : \mathbb{R} \to \mathbb{R}$ com $f(x) = \bar{U}_x(x,0,0)$, es considera l'equació:

  $$f(x) = -x + \frac{(1-\mu)(x+\mu)}{|x+\mu|^3} + \mu\frac{x-(1-\mu)}{|x-(1-\mu)|^3} = 0. \tag{3.32}$$

  La funció $f(x)$ és contínua per tot x de $A = (-\infty, -\mu) \cup (-\mu, 1-\mu) \cup (1-\mu, +\infty)$ i té dues singularitats: una a $x = -\mu$ i l'altra a $x = 1-\mu$. A més a més, se satisfà que:

  $$\begin{cases} \lim_{x\to-\infty} f(x) > 0 & \text{i} & \lim_{x\to-\mu^-} f(x) < 0 \\ \lim_{x\to-\mu^+} f(x) > 0 & \text{i} & \lim_{x\to(1-\mu)^-} f(x) < 0 \\ \lim_{x\to(1-\mu)^+} f(x) > 0 & \text{i} & \lim_{x\to+\infty} f(x) < 0 \end{cases}.$$

Pel teorema de Bolzano[14] se sap que existeix una arrel en cada interval. Afegidament, se sap que $s \mapsto \frac{s}{|s|^3}$ és estrictament decreixent, així, cada terme de $f(x)$ és estrictament decreixent on està definit. Per tant, $f'(x) < 0$ per a tot x de A, i es té que $f(x)$ és estrictament decreixent en cada interval. Es pot apreciar a la Figura 5.

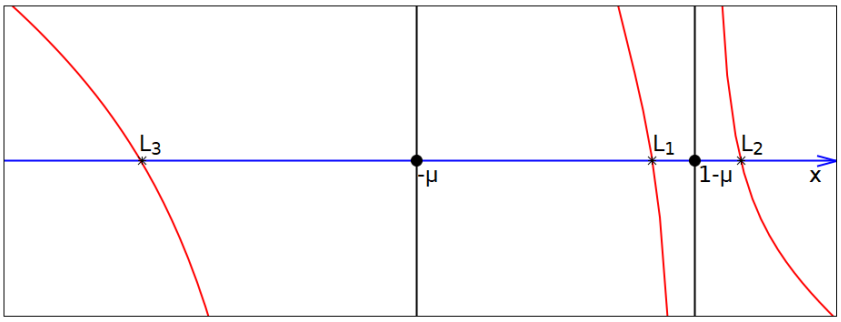


Figura 5: *Gràfica de* $f(x) = \partial_x \bar{U}(x, 0, 0)$. *Representació de la derivada del potencial respecte* x, *on s'hi aprecien els punts de Lagrange* L_1, L_2 *i* L_3.

Per trobar la solució de l'equació (3.32), es posa $u = x - 1 + \mu$. Així,

$$\begin{cases} r_1 = |x + \mu| = |u + 1| \\ r_2 = |x - 1 + \mu| = |u| \end{cases} \Rightarrow f(u+1-\mu) = -u-1+\mu+(1-\mu)\frac{(u+1)}{|u+1|^3}+\mu\frac{u}{|u|^3} = 0.$$

Posant $s_0 = signe(u)$ i $s_1 = signe(u + 1)$, s'obté

$$-u - 1 + \mu + \frac{1-\mu}{(u+1)^2}s_1 + \frac{\mu}{u^2}s_0 = 0 \tag{3.33}$$

Les possibles combinacions de la parella $(s_0, s_1) \in \{-1, 1\} \times \{-1, 1\}$ són:

$$(s_0, s_1) = (-1, -1) \Leftrightarrow \begin{cases} u + 1 < 0 \\ u < 0 \end{cases} \Leftrightarrow \begin{cases} x + \mu < 0 \\ x < 1 - \mu \end{cases} \Leftrightarrow x < -\mu$$

$$(s_0, s_1) = (-1, +1) \Leftrightarrow \begin{cases} u < 0 \\ u + 1 > 0 \end{cases} \Leftrightarrow \begin{cases} x < 1 - \mu \\ x > -\mu \end{cases} \Leftrightarrow x \in (-\mu, 1 - \mu)$$

$$(s_0, s_1) = (+1, +1) \Leftrightarrow \begin{cases} u > 0 \\ u + 1 > 0 \end{cases} \Leftrightarrow \begin{cases} x > 1 - \mu \\ x > -\mu \end{cases} \Leftrightarrow x > 1 - \mu$$

$$(s_0, s_1) = (+1, -1) \Leftrightarrow \begin{cases} u > 0 \\ u + 1 < 0 \end{cases} \Leftrightarrow \begin{cases} x > 1 - \mu \\ x < -\mu \end{cases} \Leftrightarrow x \in (1 - \mu, -\mu)$$

Aquest últim cas, on $(s_0, s_1) = (+1, -1)$, no és possible perquè $0 < \mu \leq 0.5$, per tant, s'obvia. Així doncs, si $y = 0$, tenim 3 punts d'equilibri col·lineals, resultat que concorda amb el fet que només hi havia tres solucions:

$$\begin{cases} L_3 : (s_0, s_1) = (-1, -1) & \text{si } x < -\mu \\ L_1 : (s_0, s_1) = (-1, +1) & \text{si } x \in (-\mu, 1 - \mu) \\ L_2 : (s_0, s_1) = (+1, +1) & \text{si } x > 1 - \mu \end{cases} . \tag{3.34}$$

[14] Bernard Placidus Johann Gonzal Nepomuk Bolzano (1781-1848): matemàtic, filòsof i tecnòleg txec.
Teorema de Bolzano. Sigui $f(x)$ una funció contínua en un interval $[a, b]$ tal que $f(a)f(b) < 0$, aleshores existeix un c pertanyent a l'interval (a, b) pel qual $f(c) = 0$.

De (3.33),

$$\mu\big(u^4+2u^3+u^2(1-s_1+s_0)+2us_0+s_0\big)=u^2(1+3u+3u^2+u^3-s_1),$$

i, amb això, s'obté l'equació que ha de complir la component x de cadascuna de les 3 solucions, on $x=u+1-\mu$:

$$\begin{cases} L_3: \mu(u^4+2u^3+u^2-2u-1)=u^2(u^3+3u^2+3u+2) \\ L_1: \mu(u^4+2u^3-u^2-2u-1)=u^2(u^3+3u^2+3u) \\ L_2: \mu(u^4+2u^3+u^2+2u+1)=u^2(u^3+3u^2+3u) \end{cases}. \tag{3.35}$$

Observació 3.5. Aquests polinomis de cinquè grau es coneixen com a quíntiques d'Euler. De la teoria de Galois, se sap que no existeix una fórmula algebraica per a obtenir les arrels d'un polinomi de grau més gran a 4.

Fent servir la μ del sistema Terra-Lluna ($\mu=0.012151$) i mètodes numèrics per trobar la solució (en aquest cas s'ha fet servir el mètode de Newton unidimensional), s'obtenen els 3 punts d'equilibri anomenats **col·lineals** (o *punts d'Euler*):

$$\begin{cases} L_1: (+0.836913,\ 0,\ 0, +0.836913,\ 0,\ 0) \\ L_2: (+1.155684,\ 0,\ 0, +1.155684,\ 0,\ 0) \\ L_3: (-1.013155,\ 0,\ 0, -1.013155,\ 0,\ 0) \end{cases}, \tag{3.36}$$

on s'ha tingut en compte que, per (3.28), $p_y=x$.

- **Punts triangulars.** En seran dos, es considera

$$\boxed{y\neq 0},\ \text{d'on}\ \frac{1-\mu}{r_1^3}+\frac{\mu}{r_2^3}=1.$$

Així,

$$\begin{aligned} \partial_x\bar{U} &= -x+\frac{(1-\mu)(x+\mu)}{r_1^3}+\frac{\mu(x-(1-\mu))}{r_2^3}= \\ &= -x+\frac{(x+\mu)(1-\mu)}{r_1^3}+\frac{\mu}{r_2^3}(x+\mu)-\frac{\mu}{r_2^3} \\ &= -x-\frac{\mu}{r_1^3}+(x+\mu)\Big(\frac{1-\mu}{r_2^3}+\frac{\mu}{r_2^3}\Big)=0. \end{aligned}$$

I, tenint en compte que $\frac{1-\mu}{r_2^3}+\frac{\mu}{r_2^3}=1$, i que $\mu\neq 0$:

$$-x+(x+\mu)\cdot 1-\frac{\mu}{r_2^3}=-x+x+\mu-\frac{\mu}{r_2^3}=0\Rightarrow 1-\frac{1}{r_2^3}=0\Rightarrow \boxed{r_1^3=1}$$

$$\frac{1-\mu}{r_1^3}+\frac{\mu}{r_2^3}-1=1-\mu+\frac{\mu}{r_2^3}-1=0\Rightarrow -1+\frac{1}{r_2^3}=0\Rightarrow \boxed{r_2=1}$$

Per tant, com que $r_1^3=r_2^3=1$, els punts triangulars es troben en els vèrtexs de dos triangles de costat unitari:

$$\begin{cases} 1=r_1^2=(x+\mu)^2+y^2+z^2 \\ 1=r_2^2=(x-(1-\mu))^2+y^2+z^2 \end{cases}\xrightarrow{z=0}\begin{cases} (1): & 1=(x+\mu)^2+y^2 \\ (2): & 1=(x-(1-\mu))^2+y^2 \end{cases}$$

Restant la segona equació a la primera, s'obté:

$$(x-1+\mu)^2-(x+\mu)^2=0, \tag{3.37}$$

resolent-la:

$$x=\frac{1-2\mu}{2}=\frac{1}{2}-\mu, \tag{3.38}$$

es troba la y:

$$1=r_1^2=\left((\frac{1}{2}-\mu)+\mu\right)^2+y^2=\frac{1}{4}+y^2\Leftrightarrow 1-\frac{1}{4}=y^2=\frac{3}{4}\Rightarrow \boxed{y=\pm\tfrac{\sqrt{3}}{2}}\,.$$

Així, i recordant que els punts són de la forma $(x,-y,y,x,z,0)$, s'obtenen els dos punts d'equilibri **triangulars** (també coneguts com *punts de Lagrange*) que, com s'aprecia a la Figura 6, són triangles equilàters:

$$\begin{cases} L_4:\left(\frac{1}{2}-\mu,-(+\frac{\sqrt{3}}{2}),+\frac{\sqrt{3}}{2},\frac{1}{2}-\mu,0,0\right) \\ L_5:\left(\frac{1}{2}-\mu,-(-\frac{\sqrt{3}}{2}),-\frac{\sqrt{3}}{2},\frac{1}{2}-\mu,0,0\right) \end{cases}. \tag{3.39}$$

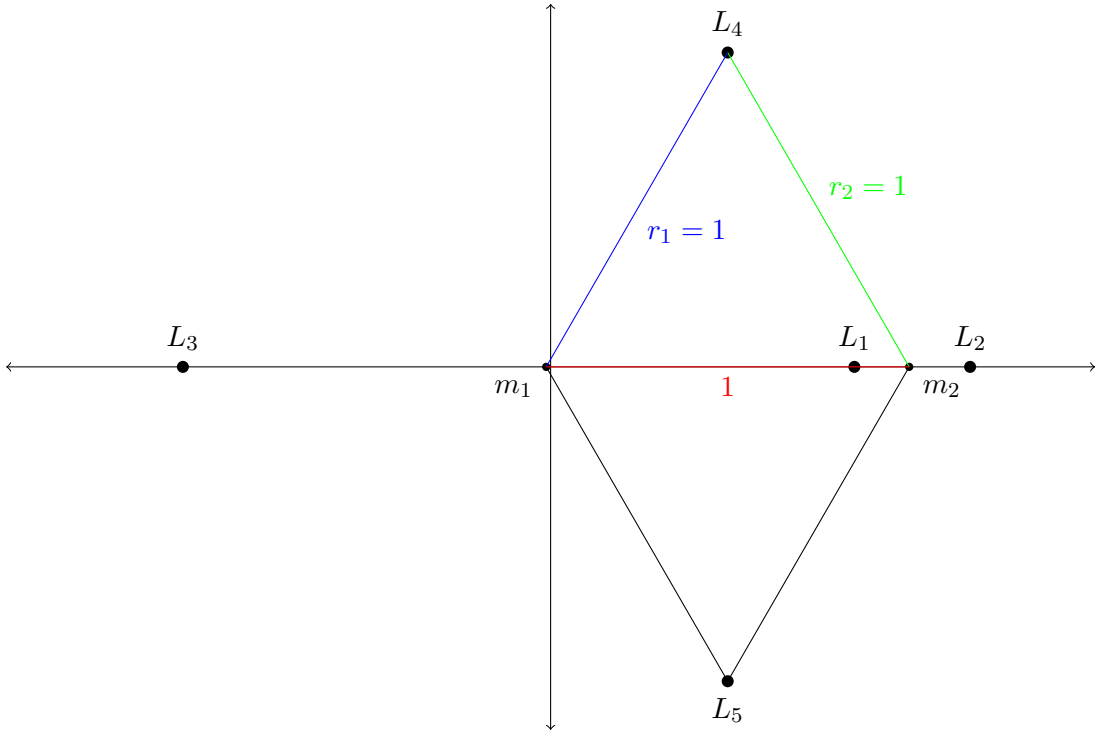


Figura 6: Localització dels cinc punts d'equilibri, amb la μ del sistema Terra-Lluna.

3.5 Estabilitat dels punts d'equilibri

En aquesta secció s'estudiarà l'estabilitat dels punts d'equilibri. Per fer-ho, es farà servir la matriu diferencial del sistema (3.27):

$$DX(\mathbf{x}) = \left(\begin{array}{cccc|cc} 0 & 1 & 1 & 0 & 0 & 0 \\ -1-\partial_{xx}\bar{U} & 0 & -\partial_{xy}\bar{U} & 1 & -\partial_{xz}\bar{U} & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ -\partial_{yx}\bar{U} & -1 & -1-\partial_{yy}\bar{U} & 0 & -\partial_{yz}\bar{U} & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 1 \\ -\partial_{zx}\bar{U} & 0 & -\partial_{zy}\bar{U} & 0 & -\partial_{zz}\bar{U} & 0 \end{array}\right). \tag{3.40}$$

On les derivades de segon ordre de $\bar{U}(x,y,z) = -\frac{1}{2}(x^2+y^2) - \frac{1-\mu}{r_1} - \frac{\mu}{r_2}$ són:

$$\left\{\begin{array}{l} \partial_{xx}\bar{U} = -1 + \frac{(1-\mu)}{r_1^3} + \frac{\mu}{r_2^3} - 3[(1-\mu)\frac{(x+\mu)^2}{r_1^5} - \mu\frac{(x-(1-\mu))^2}{r_2^5}] \\ \partial_{xy}\bar{U} = -3y[\frac{(1-\mu)(x+\mu)}{r_1^5} + \mu\frac{x-(1-\mu)}{r_2^5}] = \bar{\partial}_{yx}\bar{U} \\ \partial_{xy}\bar{U} = -3z[\frac{(1-\mu)(x+\mu)}{r_1^5} + \mu\frac{x-(1-\mu)}{r_2^5}] = \bar{\partial}_{zx}\bar{U} \\ \partial_{yy}\bar{U} = -1 + \frac{1-\mu}{r_1^3} + \frac{\mu}{r_2^3} - 3y^2[\frac{1-\mu}{r_1^5} + \frac{\mu}{r_2^5}] \\ \partial_{yz}\bar{U} = -3yz(\frac{1-\mu}{r_1^5} + \frac{\mu}{r_2^5}) = \bar{U}_{zy} \\ \partial_{zz}\bar{U} = \frac{1-\mu}{r_1^3} + \frac{\mu}{r_2^3} - 3z^2(\frac{1-\mu}{r_1^5} + \frac{\mu}{r_2^5}) = \partial_{zz}U \end{array}\right. . \tag{3.41}$$

En la secció (3.4), s'ha vist que en l'equilibri del sistema sempre es compleix que $z = 0$, d'on $\partial_{xz}\bar{U} = \partial_{zy}\bar{U} = 0$. Així doncs, $DX(\mathbf{x})$ és una matriu diagonal a blocs i, per fer el càlcul de VAPs i VEPs, es pot desacoblar la matriu diferencial en dues: una provinent de (z, p_z) i l'altra de (x, p_x, y, p_y):

$$DX_1(\mathbf{x}) = \begin{pmatrix} 0 & 1 \\ -\partial_{zz}U & 0 \end{pmatrix}; \qquad DX_2(\mathbf{x}) = \begin{pmatrix} 0 & 1 & 1 & 0 \\ -1-\partial_{xx}\bar{U} & 0 & -\partial_{xy}\bar{U} & 1 \\ -1 & 0 & 0 & 1 \\ -\partial_{yx}\bar{U} & -1 & -1-\partial_{yy}\bar{U} & 0 \end{pmatrix}$$

Observació 3.6. En el cas del Hamiltonià, la diferencial té unes simetries com, per exemple, que els VAPs van en parelles amb el seu oposat, essent també un VAP.

Per estudiar l'estabilitat dels punts fixos del sistema, es necessitarà estudiar els VAPs:

- En primer lloc, es consideren les components veritcals:

$$DX_1(\mathbf{x}) = \begin{pmatrix} 0 & 1 \\ -\partial_{zz}\bar{U} & 0 \end{pmatrix} \Rightarrow \lambda_\pm = \pm\sqrt{-\partial_{zz}\bar{U}}. \tag{3.42}$$

Com que $1 - \frac{\mu}{r_1^3},\ r_1^3,\ \mu,\ r_2^3 > 0$, i $z = 0$, tenim que $\partial_{zz}\bar{U} = \frac{1-\mu}{r_1^3} + \frac{\mu}{r_2^3} > 0$, per tot (x, y) de $\mathbb{R}^2$; de fet, en els punts d'equilibri triangulars $\partial_{zz}\bar{U} = 1$. Aleshores, dos VAPs del sistema seran $\lambda_\pm = \pm i\beta$, on $\beta = \sqrt{\partial_{zz}\bar{U}}$. Els VAPs són dos imaginaris purs. Així doncs, al ser X_1 un sistema lineal 2-dimensional, es pot concloure que per a cada punt fix les direccions verticals corresponents a X_1 són centres lineals.

Trobar els VEPs de $\lambda_\pm$ és senzill:

$$\lambda_+ = +\mathbf{i}\beta : \begin{pmatrix} -\mathbf{i}\beta & 1 \\ -\beta^2 & -\mathbf{i}\beta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \Rightarrow y = +x\mathbf{i}\beta \Rightarrow \begin{pmatrix} 1 \\ \mathbf{i}\beta \end{pmatrix} \text{ és el VEP del VAP } \lambda_+.$$

$$\lambda_- = -\mathbf{i}\beta : \begin{pmatrix} \mathbf{i}\beta & 1 \\ -\beta^2 & \mathbf{i}\beta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \Rightarrow y = -x\mathbf{i}\beta \Rightarrow \begin{pmatrix} 1 \\ -\mathbf{i}\beta \end{pmatrix} \text{ és el VEP del VAP } \lambda_-.$$

- En segon lloc, es consideraran les components horitzontals i se'n trobaran els VAPs:

$$DX_2(\mathbf{x}) = \begin{pmatrix} 0 & 1 & 1 & 0 \\ -1-\partial_{xx}\bar{U} & 0 & -\partial_{xy}\bar{U} & 1 \\ -1 & 0 & 0 & 1 \\ -\partial_{yx}\bar{U} & -1 & -1-\partial_{yy}\bar{U} & 0 \end{pmatrix}. \tag{3.43}$$

$$\begin{vmatrix} -\lambda & 1 & 1 & 0 \\ -1-\partial_{xx}\bar{U} & -\lambda & -\partial\bar{U}_{xy} & 1 \\ -1 & 0 & -\lambda & 1 \\ -\partial_{yx}\bar{U} & -1 & -1-\partial_{yy}\bar{U} & -\lambda \end{vmatrix} = \lambda^4+\lambda^2(4+\partial_{yy}\bar{U}+\partial_{xx}\bar{U})+(\partial_{xx}\bar{U}\cdot\partial_{yy}\bar{U}-(\partial_{xy}\bar{U})^2) = 0.$$

Per resoldre aquesta equació biquadrada, es pot posar $t = \lambda^2$, $B = 2 + \frac{\partial_{yy}\bar{U}+\partial_{xx}\bar{U}}{2}$ i $C = \partial_{xx}\bar{U}\cdot\partial_{yy}\bar{U} - (\partial_{xy}\bar{U})^2$, així s'obté

$$t^2 + 2Bt + C = 0 \Rightarrow t = -B \pm \sqrt{B^2 - C} \Rightarrow \boxed{\lambda = \pm\sqrt{-B \pm \sqrt{B^2 - C}}} \tag{3.44}$$

Per estudiar l'estabilitat dels punts fixos i classificar-los, també es farà distinció entre punts col·lineals i punts triangulars:

- **Punts col·lineals.** Els punts col·lineals es troben en l'eix X ($z = y = 0$), complint:

$$\begin{cases} r_1 = \sqrt{(x+\mu)^2 + y^2 + z^2} = |x+\mu| \\ r_2 = \sqrt{(x-(1-\mu))^2 + y^2 + z^2} = |x-1+\mu| \end{cases}$$

  Tenint en compte que $\frac{1-\mu}{r_1^2} = r_1 - \mu - \frac{\mu}{r_2^2}$ de $r_2 = x - 1 + \mu = r_1 - 1$, i que $C = \partial_{xx}\bar{U}\cdot\partial_{yy}\bar{U} - (\partial_{xy}\bar{U})^2 = \partial_{xx}\bar{U}\cdot\partial_{yy}\bar{U}$ ja que $\partial_{xy}\bar{U} = 0$, es veu que $B^2 - C > 0$:

$$\begin{aligned} \partial_{xx}\bar{U} &= -1 + \frac{(1-\mu)}{r_1^3} + \frac{\mu}{r_2^3} - \frac{3(1-\mu)(x+\mu)^2}{(x+\mu)^5} - 3\mu\frac{(x-1+\mu)^2}{(x-1+\mu)^5} \\ &= -1 + \frac{(1-\mu)}{r_1^3} + \mu - \frac{3(1-\mu)}{r_2^3} - 3\mu\frac{1}{r_2^3} = -1 - \frac{2(1-\mu)}{r_1^3} - \frac{\mu}{r_2^3} < 0. \\ \partial_{yy}\bar{U} &= -1 + \frac{(1-\mu)}{r_1^3} + \frac{\mu}{r_2^3} + 0 = -1 + \frac{1}{r_1}\Big(\frac{1-\mu}{r_1^2}\Big) + \frac{\mu}{r_2^3} \\ &= -1 + \frac{1}{r_1}\Big(r_1 - \mu - \frac{\mu}{r_2^2}\Big) + \frac{\mu}{r_2^3} = \frac{-\mu}{r_1} - \frac{\mu}{r_1 r_2^2} + \frac{\mu}{r_2^3} = \frac{\mu}{r_1}\Big(-1 - \frac{1}{r_2^2} + \frac{r_1}{r_2^3}\Big) \\ &= \frac{\mu}{r_1}\Big(-1 - \frac{1}{r_2^2} + \frac{1+r_2}{r_2^3}\Big) = \frac{\mu}{r_1}\Big(-1 + \frac{1}{r_2^3}\Big) > 0. \end{aligned}$$

Així, $C = \partial_{xx}\bar{U} \cdot \partial_{yy}\bar{U} < 0$. Aleshores, $B^2 - C > 0$ i $\sqrt{B^2 - C}$ és un nombre real per a tots els punts col·lineals (L_1, L_2 i L_3) i, recordant que $\lambda = \pm\sqrt{-B \pm \sqrt{B^2 - C}}$:

· Per $-B + \sqrt{B^2 - C} > 0$, tenim una parella hiperbòlica real:

$$\lambda = \pm\sqrt{-B+\sqrt{B^2 - C}}. \tag{3.45}$$

· Per $-B - \sqrt{B^2 - C} < 0$, tenim una parella el·líptica (d'imaginaris purs):

$$\lambda = \pm\sqrt{-B-\sqrt{B^2 - C}}. \tag{3.46}$$

A continuació es demostra per què passa això:
Es defineix $|B_\varepsilon| := \sqrt{B^2 - C} > 0$. S'acabem de veure que la desigualtat es compleix. I, trivialment, $|B_\varepsilon| = \sqrt{B^2 - C} > |B|$. Aleshores,

$$\text{Si } B > 0 \Rightarrow \begin{cases} -B + |B_\varepsilon| > -B + B = 0 \\ -B - |B_\varepsilon| < -B - |B| < -2B < 0 \end{cases},$$

$$\text{i si } B < 0 \Rightarrow \begin{cases} -B + |B_\varepsilon| > -B - B = -2B > 0 \\ -B - |B_\varepsilon| < -B + B = 0 \end{cases}.$$

En ambdós casos hi ha una parella hiperbòlica real i una parella el·líptica. I, afegint-li el provinent de (z, p_z), que és un centre, s'obté que els punts són

$$centre \times centre \times centre.$$

- **Punts triangulars.** Els punts triangulars (o de Lagrange) són:

$$\begin{cases} L_4 : \left(\frac{1}{2} - \mu, -(+\frac{\sqrt{3}}{2}), +\frac{\sqrt{3}}{2}, \frac{1}{2} - \mu, 0, 0\right) \\ L_5 : \left(\frac{1}{2} - \mu, -(-\frac{\sqrt{3}}{2}), -\frac{\sqrt{3}}{2}, \frac{1}{2} - \mu, 0, 0\right) \end{cases}. \tag{3.47}$$

Anàlogament, avaluant els VAPs en aquests punts, s'obté l'estabilitat d'aquests. Recordant que $r_1 = r_2 = 1$:

$$\begin{aligned} \partial_{yx}\bar{U}(L_4) &= -3y[(1-\mu)(\frac{1}{2} - \mu + \mu) + \mu\frac{\frac{1}{2} - \mu - 1 + \mu}{1}]\Big|_{y=\frac{\sqrt{3}}{2}} = -3\frac{\sqrt{3}}{2}\frac{1}{2}[1 - \mu - \mu] \\ &= -\frac{\sqrt{27}}{4}(1 - 2\mu). \\ \partial_{yx}\bar{U}(L_5) &= \frac{\sqrt{27}}{4}(1 - 2\mu). \end{aligned}$$

Així, $(\partial_{yx}\bar{U})^2(L_4) = (\partial_{yx}\bar{U})^2(L_5)$.

$$\begin{aligned} \partial_{yy}\bar{U}(L_4) &= -1 + 1 - \mu + \mu - 3\frac{3}{4}(1 - \mu + \mu) = -\frac{9}{4} = \partial_{yy}\bar{U}(L_5). \\ \partial_{xx}\bar{U}(L_4) &= -1 + 1 - \mu + \mu + (1-\mu)(\frac{1}{2} + \mu - \mu)^2(-3) + \mu(\frac{1}{2} - \mu - 1 + \mu)^2(-3) \\ &= (1-\mu)\frac{-3}{4} + \mu\frac{-3}{4} = \frac{-3}{4} = \partial_{xx}\bar{U}(L_5). \end{aligned}$$

Aleshores, per $i \in \{4,5\}$,

$$\begin{aligned} C(L_i) &= \partial_{xx}\bar{U}\cdot\partial_{yy}\bar{U} - (\partial_{xy}\bar{U})^2 = (\frac{-3}{4})(\frac{-9}{4}) - \frac{27}{16}(1-2\mu)^2 \\ &= \frac{27}{16} - \frac{27}{16}(1-4\mu+4\mu^2) = \frac{27}{16}(4\mu-\mu^2) = \frac{27}{4}\mu(1-\mu). \\ B(L_i) &= 2 + \frac{(\frac{-9}{4}) - (\frac{3}{4})}{2} = 2 + \frac{-12}{8} = 2 - \frac{3}{2} = \frac{1}{2}. \end{aligned}$$

Per tant, els VAPs en L_4 i L_5 són:

$$\begin{aligned} \lambda &= \pm\sqrt{-B \pm \sqrt{B^2 - C}} = \pm\sqrt{-\frac{1}{2} \pm \sqrt{\frac{1}{4} - \frac{27}{4}\mu(1-\mu)}} \\ &= \pm\sqrt{-\frac{1}{2} \pm \frac{1}{2}\sqrt{1-27\mu(1-\mu)}} = \pm\sqrt{\frac{1}{2}\big(-1 \pm \sqrt{1-27\mu(1-\mu)}\ \big)}\ . \end{aligned}$$

Si $1 - 27\mu(1-\mu) \geq 0$, les arrels seran purament imaginàries (parella el·líptica). Altrament, es tindrà una parella hiperbòlica real.

Considerant la igualtat

$$1 - 27\mu(1-\mu) = 0 \Leftrightarrow 27\mu^2 - 27\mu + 1 = 0,$$

s'obtenen les solucions

$$\mu_\pm = \frac{27 \pm \sqrt{27^2 - 4\cdot 27}}{2\cdot 27} = \frac{1}{2}(1 \pm \sqrt{\frac{23}{27}}) \simeq \begin{cases} \mu_+ = 0.961 \\ \mu_- = 0.038521 \end{cases}\ .$$

Com que s'està considerant que $0.5 \geq \mu\ > 0$ i $\mu_+ \simeq 0.961 > 0.5$, μ_+ és incompatible i, per tant, la solució és μ_-.

Així doncs, per a $\mu < \mu_- = \frac{1}{2}(1 - \sqrt{\frac{23}{27}}) = \mu_R \simeq 0.03852...$, que és el **paràmetre de Routh**, es conclou que L_4 i L_5 són dos centres i, agefint-li el provinent de (z, p_z), que també és un centre, els punts són:

$$centre \times centre \times centre.$$

I, per $\mu > \mu_R$ es tindria:

$$centre \times sella\ complexa.$$

En el cas del sistema Terra-Lluna, el que s'està estudiant, com que $\mu = 0.012151 < \mu_R$, L_4 i L_5 són:

$$centre \times centre \times centre.$$

4 Òrbites periòdiques

El problema que es planteja en aquest treball és calcular òrbites periòdiques (a través de mètodes de Fourier) d'un sistema diferencial d'equacions diferencials ordinàries (EDOs) amb almenys una integral primera, com és el cas d'un sistema Hamiltonià; on tenim que si $H = H(\mathbf{q}, \mathbf{p})$ és independent del temps, aleshores H és una integral primera del sistema Hamiltonià.

Per trobar on són les solucions periòdiques d'un sistema d'EDOs amb almenys una integral primera pròxima a la solució periòdica del sistema lineal (com pot ser un sistema Hamiltonià), es fa servir el teorema de Lyapunov[15].

4.1 Anàlisi de les linealitzacions al voltant dels punts fixos

A l'estudi dels sistemes dinàmics no lineals, una tècnica comuna és analitzar el comportament del sistema al voltant dels punts fixos (o d'equilibri) a través de la linealització d'aquests sistemes. Aquest mètode aproxima el sistema no lineal per un sistema lineal a prop del punt fix. L'aproximació es realitza utilitzant la matriu diferencial del sistema en aquest punt fix.

Quan es realitza la linealització d'un sistema dinàmic, els punts fixos poden presentar diferents tipus d'estabilitat, determinada pels valors propis de la matriu diferencial (com ja s'ha vist en les seccions anteriors). Les **components centrals** fan referència als modes associats als valors propis imaginaris purs (amb part real zero), indicant que el sistema presenta un comportament oscil·latori al voltant del punt fix.

Els VAPs imaginaris suggereixen que existeixen solucions periòdiques al voltant del punt fix. Aquests modes oscil·latoris centrals poden ser representats per combinacions lineals dels VEPs associats a aquests VAPs imaginaris. Així, cada component central de la linealització dona lloc a una família de solucions periòdiques que oscil·len, amb una freqüència determinada, pels VAPs. Per trobar aquestes solucions periòdiques, es poden utilitzar els VEPs de la matriu diferencial.

Suposem, doncs, que tenim un sistema dinàmic que ha estat linealitzat amb valors propis imaginaris $\mp i\omega$. En estudiar les linealitzacions al voltant dels punts fixos d'un sistema dinàmic s'obté, pel teorema de Lyapunov (que s'enunciarà en 4.2), que els components centrals associats als VAPs imaginaris indiquen l'existència de solucions periòdiques. Sigui $\mathbf{w} = \mathbf{u} + \mathbf{iv}$ és el VEP de $-\mathbf{i}\omega$, aleshores $\mathbf{u}$ i $\mathbf{v}$ es poden utilitzar per a construir solucions periòdiques amb parametritzacions temporals de la forma

$$\mathbf{x}(t) = r\big(\cos(\omega t)\mathbf{u} - \sin(\omega t)\mathbf{v}\big), \tag{4.1}$$

on A i B són constants determinades per les condicions inicials del sistema. Aquesta forma parametritza solucions periòdiques en termes del temps t del sistema linealitzat.

La freqüència ω de les solucions periòdiques ve determinada pels VAPs imaginaris: si els VAPs són $\mp i\omega$, la freqüència de les oscil·lacions periòdiques al voltant del punt fix és ω.

[15] Aleksandr Mikhàilovitx Lyapunov (1857-1918): matemàtic rus, mecànic i físic que va treballar en EDOs, mecànica celeste i teoria de la probabilitat.

4.2 Persistència de les famílies d'òrbites periòdiques en sistemes Hamiltonians

Un aspecte fonamental sobre els sistemes dinàmics Hamiltonians és que tenen famílies d'òrbites periòdiques. Això és degut a les seves propietats intrínsiques de conservació i les propietats d'estructura matemàtica, com ara la conservació de l'energia i l'existència d'integrals primeres, juntament amb teoremes com el de Poincaré[16]-Birkhoff[17].

Un altre aspecte destacable dels sistemes dinàmics Hamiltonians és que les famílies d'òrbites periòdiques identificades mitjançant la linealització al voltant dels punts fixos persisteixen quan considerem el sistema complet (no lineal). Aquesta característica és fonamental per comprendre la dinàmica global del sistema i es conclou al final de la secció.

El **Teorema de Lyapunov per a òrbites periòdiques** implica que en un sistema Hamiltonià, les òrbites periòdiques observades en la linealització al voltant dels punts fixos també es mantenen en el sistema original no lineal.

Abans d'enunciar el teorema de Lyapunov, es necessita d'una definició:

Definició 4.1. *Sigui $\dot{\mathbf{x}} = f(\mathbf{x})$ un sistema d'equacions diferencials i sigui $\Omega \subset \mathbb{R}^n$ l'espai de fase[18], es considera una funció $g : \Omega \to \mathbb{R}$ no localment constant amb derivades contínues en Ω. Es diu que $g(\mathbf{x})$ és **integral primera de** $\dot{\mathbf{x}} = f(\mathbf{x})$ si, per a tot $\mathbf{x}$,*

$$Dg(\mathbf{x})f(\mathbf{x}) = 0, \tag{4.2}$$

fet que implica que g sigui constant sobre les solucions del sistema.

L'existència d'una integral primera implica l'existència de famílies d'òrbites periòdiques que neixen de punts fixos que tenen VAPs centrals sota condicions de no-ressonància.

Teorema 4.2 (Lyapunov)**.** *Considerem que el sistema n-dimensional autònom*

$$\dot{\mathbf{x}} = A\mathbf{x} + f(\mathbf{x}), \tag{4.3}$$

on f és una funció diferenciable tal que, en $\mathbf{x} = \mathbf{0}$, valen zero ella i les seves primeres derivades. Suposem que el sistema admet una integral primera de la forma $I(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T S\mathbf{x} + o(|\mathbf{x}|^2)$, amb $S = S^T$ i $det(S) \neq 0$. Suposem també que els VAPs de A són de la forma $\pm\mathbf{i}\beta, \lambda_3, ..., \lambda_n$ on $\mathbf{i}\beta$ és un imaginari pur. Si $\frac{\lambda_j}{\mathbf{i}\beta} \notin \mathbb{Z}$, i.e. es compleix la condició de no- ressonància, per $j = 3 \div n$, aleshores existeix una família uniparamètrica d'òrbites periòdiques que quan tendeixen al punt d'equilibri el seu període tendeix a $T = \frac{2\pi}{\beta}$.[10]

Una demostració d'aquest teorema es troba a l'Annex A.2.

L'existència d'aquestes òrbites periòdiques depèn de que es satisfacin unes condicions de no-ressonància. Aquestes condicions asseguren que els VAPs del sistema linealitzat no

[16] Henri Poincaré (1854-1912): matemàtic francès destacat pels seus treballs sobre equacions diferencials i les seves aplicacions a la mecànica celeste.

[17] George David Birkhoff (1884-1944): matemàtic americà. Nombroses contribucions a la teoria d'EDOs, sistemes dinàmics, el problema dels 4 colors i relativitat general.

[18] L'**espai de fase** és un conjunt multidimensional en el qual cada dimensió representa una variable que descriu l'estat del sistema en un instant donat. Els punts en aquest espai corresponen als possibles estats del sistema.

són en ressonància, és a dir, garanteixen que les freqüències naturals del sistema no són múltiples enteres les unes de les altres.

En el cas que ocupa aquest treball, aquestes condicions de no-ressonància es compleixen. Per tant, es pot afirmar que les famílies d'òrbites periòdiques identificades a través de la linealització al voltant dels punts fixos persisteixen en el sistema Hamiltonià complet.

Aquest resultat és important perquè permet extrapolar les conclusions obtingudes de l'anàlisi linealitzada a l'estudi del sistema dinàmic no lineal en la seva totalitat, proporcionant una eina poderosa per a la comprensió del comportament global del sistema.

4.3 Cerca de solucions periòdiques a un nivell d'energia fixat

En el context dels sistemes dinàmics Hamiltonians, un problema clau és la cerca de solucions periòdiques a un nivell d'energia fixat. És fonamental, ja que les òrbites periòdiques juguen un paper central en la comprensió de la dinàmica del sistema.

Es considera un sistema Hamiltonià amb una funció Hamiltoniana $H(\mathbf{q}, \mathbf{p})$, on $\mathbf{q}$ i $\mathbf{p}$ són les coordenades generalitzades i els moments conjugats, respectivament. Les equacions de moviment de Hamilton són:

$$\dot{\mathbf{q}} = \frac{\partial H}{\partial \mathbf{p}}, \quad \dot{\mathbf{p}} = -\frac{\partial H}{\partial \mathbf{q}}. \tag{4.4}$$

Es volen solucions periòdiques d'aquestes equacions; és a dir, funcions $\mathbf{q}(t)$ i $\mathbf{p}(t)$ que satisfacin: $\mathbf{q}(t+T) = \mathbf{q}(t)$, $\mathbf{p}(t+T) = \mathbf{p}(t)$, per a algun període T. Aquestes solucions periòdiques es corresponen amb òrbites tancades en l'espai de fase.

Fixar un nivell d'energia vol dir que es busquen aquestes solucions periòdiques sota la restricció que la funció Hamiltoniana tingui un valor constant H_0 per a tot t:

$$H(\mathbf{q}(t), \mathbf{p}(t)) = H_0, \tag{4.5}$$

Així doncs, les incògnites són les funcions periòdiques $\mathbf{q}(t)$ i $\mathbf{p}(t)$, i la freqüència $\omega = \frac{1}{T}$. Alternativament, si es fixa la freqüència, l'energia H_0 es converteix en la incògnita. Aquest problema es pot abordar mitjançant diverses tècniques analítiques i numèriques.

La cerca de solucions periòdiques a un nivell d'energia fixat és crucial perquè permet identificar estructures regulars[19] en el sistema dinàmic i, com s'ha vist anteriorment, comprendre millor la seva dinàmica global. Les òrbites periòdiques poden actuar com a esquelets que organitzen el comportament a llarg termini del sistema i l'estudi d'aquestes òrbites proporciona informació valuosa de la natura del moviment en diferents nivells d'energia.

La cerca de solucions periòdiques a un nivell d'energia fixat ofereix una visió profunda del comportament dinàmic dels sistemes Hamiltonians, i les incògnites clau són les funcions periòdiques que descriuen les trajectòries juntament amb la freqüència d'oscil·lació o el nivell d'energia, segons el cas.

[19] Una **estructura regular** és una configuració ordenada i periòdica que es repeteix dins del sistema, com ara òrbites periòdiques i corbes invariants, que ajuden a comprendre i preveure el comportament a llarg termini del sistema.

4.4 Mètode per a trobar òrbites periòdiques

En aquest treball, donat un camp $\dot{\mathbf{x}}_P(t) = f(\mathbf{x}_P(t))$ en $\mathbb{R}^m$; en lloc de buscar el flux $\varphi(t; x_0)$ que, complint $\varphi(T, x_0) = x_0$, és solució del sistema

$$\begin{cases} \dot{\mathbf{x}}_P(t) = f(\mathbf{x}_P(t)) \\ \mathbf{x}_P(T) = \mathbf{x}_P(0) \end{cases}, \tag{4.6}$$

es busca una òrbita periòdica $k : \mathbb{T} \to \mathbb{R}^m$, on $\mathbb{T} = \mathbb{R}/\mathbb{Z}$ tal que, per a $\omega \in \mathbb{R} \setminus \{0\}$, es compleixi la següent equació

$$f(k(\theta)) = k'(\theta)\omega. \tag{4.7}$$

Vegem com $\varphi(t, k(\theta)) = k(\theta + \omega t)$ és solució del sistema:

Demostració. Per $t = 0$, $\varphi\big(0; k(\theta)\big) = k(\theta + \omega \cdot 0) = k(\theta)$.

La funció $k(\theta + \omega t)$ satisfà l'EDO si $f\big(k(\theta + \omega t)\big) = \frac{d}{dt}\big(k(\theta + \omega t)\big)$. Aquesta igualtat es compleix ja que:

$$\frac{d}{dt}\big(k(\theta + \omega t)\big) = k'(\theta + \omega t)\omega = f\big(k(\theta + \omega t)\big).$$

Així doncs, $\mathbf{x}_P(t) = \varphi(t, k(\theta))$. I, per unicitat, $\mathbf{x}_P(t) = \varphi(t, k(0)) = k(\omega t)$. □

En aquest context, la teoria de Fourier resulta especialment rellevant. La transformada de Fourier permet descompondre una funció periòdica en una sèrie sinusoidal, fet que facilita l'anàlisi de les propietats de l'òrbita periòdica $k(\theta)$. Com es veurà en el proper capítol, es pot representa $k(\theta)$ com una sèrie de Fourier

$$k(\theta) = \sum_{n=-\infty}^{\infty} c_n e^{\mathbf{i}2\pi n\theta}, \tag{4.8}$$

on c_n són els coeficients de Fourier. Això proporciona una representació que pot simplificar l'estudi de les solucions periòdiques i la seva estabilitat. En particular, l'ús de Fourier permet analitzar la resposta del sistema a diferents freqüències ω, facilitant la comprensió dels patrons de moviment en el sistema dinàmic estudiat.

5 Breu introducció a mètodes de Fourier

Fourier[20] va ser un matemàtic i físic francès que va fer, en diverses àrees de la ciència, contribucions importants: la sèrie de Fourier (representacions de funcions periòdiques com una suma infinita de funcions trigonomètriques), l'equació de conducció de la calor (propagació de la calor en un material) i la Transformada de Fourier (descompon una funció en les seves components de freqüència), entre d'altres.

En aquest treball s'han usat dues de les seves contribucions: les Sèries de Fourier i la Transformada de Fourier. Les sèries de Fourier permetran reescriure una funció periòdica com una combinació lineal infinita de funcions periòdiques base $e^{\mathbf{i}2\pi n\frac{x}{P}}$, on P representa el període de la funció periòdica.

Aquest capítol es basa, principalment, en el llibre *Fast Fourier transforms* [11].

5.1 Aspectes bàsics de les sèries de Fourier

Definició 5.1. *Sigui $(X, \mathcal{A}, \mu)$ un espai mesurable, on X és un conjunt, $\mathcal{A}$ una sigma-àlgebra de subconjunts de X, i μ una mesura sobre $(X, \mathcal{A})$. L'**espai de Lebesgue** $L^p(X, \mu)$ està format per totes les funcions mesurables $f : X \to \mathbb{C}$ tals que la p-èsima potència del valor absolut de f té una integral finita. i.e.*

$$L^p(X,\mu) = \Big\{ f : X \to \mathbb{C} \mid f \text{ és mesurable i } \|f\|_p = \left(\int_X |f(x)|^p \, d\mu(x) \right)^{1/p} < \infty \Big\}, \quad (5.1)$$

*on $\|f\|_p$ s'anomena **norma** L^p.*

Definició 5.2. *Sigui $g : \mathbb{R} \to \mathbb{R}$ una funció real en L^1 de període P, els **coeficients de Fourier** $\{c_n\}$ per g en l'interval $[0, P]$ venen definits per:*

$$c_n = \frac{1}{P} \int_0^P g(x) e^{-\mathbf{i}2\pi nx/P} dx, \text{ per tot } n \text{ enter.} \quad (5.2)$$

*Fent servir aquests coeficients $\{c_n\}$, la **sèrie de Fourier** de g ve definida per la part dreta de la següent correspondència:*

$$g \backsim \sum_{n=-\infty}^{\infty} c_n e^{\mathbf{i}2\pi nx/P} \quad (5.3)$$

Nota. No es pot posar una igualtat sense demostrar-la: es fa servir el símbol de correspondència $\backsim$. La demostració de la igualtat es pot trobar en [11].

En el cas de les funcions analítiques, els coeficients de Fourier decreixen exponencialment ràpid i la convergència de la sèrie a la funció és uniforme. Aquests coeficients venen donats per integrals que es poden aproximar via el mètode dels trapezis amb l'anomenada transformada discreta de Fourier (DFT).

[20] Jean-Baptiste-Joseph Fourier (1768-1830): matemàtic, físic i egiptòleg francès.

5.2 Transformada de Fourier Discreta. DFT

La transformada de Fourier és, conceptualment, l'espectre de freqüències d'una funció: descompon una funció temporal en les freqüències que la constitueixen.

La DFT és una operació que permet obtenir la versió mostrejada de la Transformada de Fourier d'un senyal discret. Es derivarà aproximant coeficients de Fourier: es considera el coeficient *k-èsim*, c_k, per a una funció g, fent servir exponencials de període P:

$$c_k = \frac{1}{P}\int_0^P g(x)e^{-\mathbf{i}2\pi kx/P}dx. \tag{5.4}$$

S'aproxima aquesta integral amb una suma de Riemann utilitzant la regla dels trapezis:

$$c_k \approx \frac{1}{P}\sum_{j=0}^{N-1} g(x_j)e^{-\mathbf{i}2\pi kx_j/P}\frac{P}{N}, \tag{5.5}$$

on $x_j = jP/N$ per $j = 0, 1, ..., N-1$. Per a aquest conjunt de punts $\{x_j\}_j^{N-1} = 0$,

$$c_k \approx \frac{1}{N}\sum_{j=0}^{N-1} g(j\frac{P}{N})e^{-\mathbf{i}2\pi kx/P}. \tag{5.6}$$

Definició 5.3. *Donats N nombres complexos, $\{g_j\}_{j=0}^{N-1}$, es denota la seva **DFT** de N punts com $\{\tilde{g}_k\}$, on $\tilde{g}_k$ ve definida per:*

$$\tilde{g}_k = \sum_{j=0}^{N-1} g_j e^{-i2\pi jk/N}, \tag{5.7}$$

per a tot k enter.

D'ara en endavant es farà servir la lletra ω per fer referència tant a $e^{-\mathbf{i}2\pi/N}$ com a $e^{\mathbf{i}2\pi/N}$. Així, $\omega^N = 1$, que serà l'única propietat especial de ω que necessitarà.

Teorema 5.4. *Suposem que la seqüència $\{h_j\}_{j=0}^{N-1}$ té com a DFT de N punts $\{\tilde{h}_k\}$ i la seqüència $\{g_j\}_{j=0}^{N-1}$ té com a DFT de N punts $\{\tilde{g}_k\}$. Aleshores, es compleixen les següents propietats:*

a. *Linealitat. Per a totes les constants complexes a i b, la seqüència $\{ah_j + bg_j\}_{j=0}^{N-1}$ té com a DFT de N punts a $\{a\tilde{h}_k + b\tilde{g}_k\}$.*

b. *Periodicitat. Per a tots els enters k tenim $\tilde{g}_{k+N} = \tilde{g}_k$*

c. *Inversió. Per $j = 0, 1, ..., N-1$,*

$$g_j = \frac{1}{N}\sum_{k=0}^{N-1} \tilde{g}_k e^{\mathbf{i}2\pi jk/N} \tag{5.8}$$

Demostració. Vegem que es compleixen cadascuna de les propietats:

a. **Linealitat**. Siguin a i b constants complexes,

$$\sum_{j=0}^{N-1}(ah_j+bg_j)e^{\mathbf{i}2\pi jk/N}=a\sum_{j=0}^{N-1}(h_j)e^{\mathbf{i}2\pi jk/N}+b\sum_{j=0}^{N-1}(g_j)e^{\mathbf{i}2\pi jk/N}=a\tilde{h}_k+b\tilde{g}_k$$

b. **Periodicitat**. Posem $\omega=e^{-\mathbf{i}2\pi/N}$. Sigui k un enter, la DFT $\{\tilde{g}_k\}$ ve definida per

$$\tilde{g}_k=\sum_{j=0}^{N-1}g_j\omega^{jk}.$$

Per tant,

$$\tilde{g}_{k+N}=\sum_{j=0}^{N-1}g_j\omega^{j(k+N)}=\sum_{j=0}^{N-1}g_j\omega^{jk}(\omega^N)^j=\sum_{j=0}^{N-1}g_j\omega^{jk}=\tilde{g}_k.$$

c. **Inversió**. Notem que $\omega^{-1}=e^{\mathbf{i}2\pi/N}$, aleshores $\omega^{-jk}=e^{\mathbf{i}2\pi jk/N}$. Per tant,

$$\begin{aligned}\frac{1}{N}\sum_{k=0}^{N-1}\tilde{g}_ke^{\mathbf{i}2\pi jk/N}&=\frac{1}{N}\sum_{k=0}^{N-1}\tilde{g}_k\omega^{-jk}=\frac{1}{N}\sum_{k=0}^{N-1}\left[\sum_{m=0}^{N-1}g_m\omega^{mk}\right]\omega^{-jk}\\&=\frac{1}{N}\sum_{k=0}^{N-1}\left[\sum_{m=0}^{N-1}g_m\omega^{mk}\omega^{-jk}\right].\end{aligned}$$

Tenint en compte que $\omega^{mk}\omega^{-jk}=\omega^{(m-j)k}$ i canviant l'ordre de les sumes:

$$\frac{1}{N}\sum_{k=0}^{N-1}\tilde{g}_ke^{\mathbf{i}2\pi jk/N}=\frac{1}{N}\sum_{m=0}^{N-1}g_m\left[\sum_{k=0}^{N-1}\omega^{(m-j)k}\right]$$

Per una j fixada, si $m\neq j$, $\omega^{m-j}\neq 1$. Per tant, fent servir la suma geomètrica, i tenint en compte que $\omega^N=1$, s'obté:

$$\sum_{k=0}^{N-1}\omega^{(m-j)k}=\frac{1-(W^{(m-j)})^{(N-1)+1}}{1-\omega^{(m-j)}}=\frac{0}{1-\omega^{(m-j)}}=0.$$

Si $m=j$, aleshores $\omega^{(m-j)k}=\omega^0=1$ i:

$$\frac{1}{N}\sum_{k=0}^{N-1}\tilde{g}_ke^{\mathbf{i}2\pi jk/N}=\frac{1}{N}g_j\sum_{k=0}^{N-1}1=g_j.$$

□

Observació 5.5. Una conseqüència de la propietat d'inversió és que dues seqüències diferents NO poden tenir la mateixa DFT.

Definició 5.6. *Si $\{G_k\}_{k=0}^{N-1}$ és una seqüència de N nombres complexos, la seva **DFT inversa** de N punts com $\{H_k\}$, on H_k ve definida per:*

$$g_j=\sum_{k=0}^{N-1}G_ke^{i2\pi jk/N} \tag{5.9}$$

per a tot j enter.

Observació 5.7. Posant $\omega = e^{\mathbf{i}2\pi jk/N}$, la DFT inversa té també propietats de linealitat, periodicitat i inversió. De fet, la formula d'inversió per la DFT inversa és:

$$G_k = \frac{1}{N}\sum_{j=0}^{N-1} g_j e^{-\mathbf{i}2\pi jk/N}. \tag{5.10}$$

5.3 Algorisme de Cooley-Turkey. FFT

Els anys seixanta del segle passat van ser testimonis de moments significatius com, per exemple, l'arribada de la humanitat a la Lluna o l'inici de la construcció del mur de Berlin. Durant aquest període, Cooley[21] i Tukey[22] van concebre l'algorisme de la Transformada Ràpida de Fourier (FFT). Aquesta innovació va transformar radicalment el camp del processament digital de senyals. De fet, l'article on ho van plasmar, *An algorithm for the Machine Calculation of Complex Fourier Series* [4], va ser una gran contribució per a l'anàlisi numèric i és un dels articles matemàtics més citats de la segona meitat del segle XX.

Reduir el nombre d'operacions necessàries pot reduir dràsticament el temps computacional necessari per executar un algorisme incrementant-ne l'eficiència. La DFT es pot implementar amb l'algorisme de la FFT.

Aquest algorisme optimitza la DFT amb un mecanisme astut de factorització de matrius que aprofita l'estructura peculiar de la matriu de Fourier. Un càlcul directe d'una DFT de N-punts necessita de $(N-1)^2$ multiplicacions, i $N(N-1)$ sumes: un total de $2N^2-3N+1$ operacions. La FFT redueix el nombre de multiplicacions necessàries d'ordre N^2 a ordre $N\log_2 N$.

5.4 Eliminació del terme de Nyquist i reordenament dels punts

Per a aquest treball, és fonamental entendre la relació entre la freqüència de Nyquist[23] i la periodicitat de les funcions analitzades.

Definició 5.8. *El **terme de Nyquist** fa referència a la freqüència de Nyquist, que és la meitat de la freqüència de mostreig d'un sistema discret.*

En l'algorisme del treball es vol que el terme Nyquist sigui prescindible *i.e.* igual a zero. Per això, es reordenaran els punts per tal de centrar l'espectre de freqüències centrat al voltant de zero. Al tenir una funció periòdica el recorregut de la funció es mantindrà: en el cas d'estar davant d'una discretització es tindran els mateixos coeficients reordenats.

[21] James William Cooley (1926-2016): matemàtic americà.

[22] John Wilder Tukey (1915-2000): matemàtic i estadístic americà.

[23] Harry Nyquist (1889-1976): enginyer nord-americà d'origen suec amb contribucions a la teoria de la informació.

Reordenament dels índexs de la DFT

Tradicionalment, els índexs de la DFT es consideren de 0 a $N-1$. Això inclou el terme de Nyquist a $k = N/2$, per N parell. En interpretar-se com a (aproximacions de) coeficients de Fourier, per analitzar millor la simetria de les freqüències, es poden reordenar els índexs per anar de $-N/2$ a $N/2-1$ (considerant N parell), eliminant així el terme de Nyquist explícitament.

- Primer, es divideixen els coeficients en dos grups: de g_0 a $g_{N/2-1}$ i de $g_{N/2}$ a g_{N-1}.
- Després, es reordenen els índexs perquè $g_{N/2}$ a g_{N-1} es moguin a les posicions negatives: $g_{N/2}$ es mou a $g_{-N/2}$, $g_{N/2+1}$ a $g_{-N/2+1}$, i així successivament.

Matemàticament, aquesta reordenació es pot expressar com:

$$g_k = \begin{cases} g_{k+N}, & \text{si } k < 0 \\ g_k, & \text{si } 0 \leq k < N/2 \\ g_{k-N}, & \text{si } k \geq N/2 \end{cases}$$

Així, en lloc de treballar amb els punts de freqüència de 0 a $N-1$, es treballa amb els punts de $-N/2$ a $N/2-1$. Aquest reordenament fa que l'anàlisi de les funcions periòdiques sigui més intuïtiva, ja que centra l'espectre al voltant de la freqüència zero, permetent una interpretació més clara de les components de freqüència positives i negatives.

Aquesta reordenació també és coneguda com a "***FFT shift***" i és implementada en moltes biblioteques de programari per a facilitar l'anàlisi de senyals en l'àmbit de les freqüències. Així doncs, en aquest treball s'eliminarà el terme de Nyquist i s'utilitzarà el rang de punts de $-N/2$ a $N/2-1$ a l'hora de calcular la derivada, que es necessitarà per al càlcul d'òrbites periòdiques buscat.

6 Algorisme de càlcul d'òrbites periòdiques amb Fourier

L'objectiu final d'aquest treball és calcular òrbites periòdiques i, per fer-ho, s'utilitzarà teoria de Fourier. En aquest capítol s'explica una estratègia per a calcular-les. D'ara en endavant, els sistemes seran tan diferenciables com es necessiti.

6.1 Plantejament del problema

Donat un camp $\dot{\mathbf{x}} = X(\mathbf{x})$ en $\mathbb{R}^m$ amb $\varphi = \varphi(t;\mathbf{x})$ com a flux, es volen trobar òrbites periòdiques parametritzades per

$$k : \mathbb{T} = \mathbb{R}/\mathbb{Z} \longrightarrow \mathbb{R}^m. \tag{6.1}$$

De manera que, amb ω com a freqüència de l'òrbita k, es compleixi

$$X(k(\theta)) = k'(\theta)\omega. \tag{6.2}$$

Així doncs, com ja s'ha vist a 4.4, com que $\frac{d}{dt}\big(k(\theta+\omega t)\big) = k'(\theta+\omega t)\cdot\omega = X(k(\theta+\omega t))$, i suposant que la solució $\varphi(t;k(0)) = k(\omega t)$ és una funció periòdica de període $T = \frac{1}{\omega}$. Aleshores, com que $\varphi(0,k(0)) = k(0)$, s'obté que $k(\theta+\omega t)$ satisfà el Problema de Cauchy

$$\begin{cases} \dot{\mathbf{x}} = X(\mathbf{x}) \\ \mathbf{x}(0) = k(0) \end{cases}, \tag{6.3}$$

i, per unicitat, $k(\omega t) = \varphi\big(t;k(0)\big)$.

Les solucions buscades en aquest problema seran òrbites periòdiques descrites, trivialment, per funcions periòdiques. Així doncs, per manipular-les, es farà ús de la teoria de Fourier.

Suposant que X té integral primera H, es fixarà el punt inicial de l'òrbita periòdica, $k(0)$, sobre una **secció transversal** al camp en el punt $k(0)$. Aquesta ve donada per una equació de la forma $g(x) = g_0$, on $g : \mathbb{R}^m \to \mathbb{R}$. Així, $g(k(0)) = g_0$, i $Dg(k(0))X(k(0)) \neq 0$.

Per aconseguir $X(k(\theta)) - k'(\theta)\omega = 0$, es tindrà en compte que X té integral primera, aleshores, per fixar una lligadura entre H i ω, s'afegeix la condició que la solució periòdica de període $T = \frac{1}{\omega}$ estigui a un cert nivell d'energia. Així doncs, la freqüència depèn de l'energia ($\omega = \omega(H)$). Fixant $H_0 = H(K(0))$ es buscarà ω i k.

Observació 6.1. Per trobar òrbites periòdiques planes per als punts col·lineals, on $z = 0$, es pot considerar que l'òrbita comença en $y = 0$. Així doncs, si es tingués l'òrbita definida per k amb freqüència ω, es compliria, pel funcional

$$\begin{aligned} G : A(\mathbb{T},\mathbb{R}^m)\times\mathbb{R} &\longrightarrow A(\mathbb{T},\mathbb{R}^m)\times\mathbb{R}^2. \\ (\bar{k},\hat{\omega}) &\longmapsto G(\bar{k},\hat{\omega}) = \begin{pmatrix} X(\bar{k}(\theta)) - \bar{k}'(\theta)\hat{\omega} \\ H(\bar{k}(0)) - H_0 \\ g(\bar{k}(0)) - g_0 \end{pmatrix}, \end{aligned} \tag{6.4}$$

que $G(k,\omega) = 0$.

Partint d'una aproximació de l'òrbita, s'obtindria

$$G(k,\omega) = (E(\theta), e_H, e_g)^T, \tag{6.5}$$

on $E : \mathbb{T} \to \mathbb{R}^m$, e_H i e_g representen els errors del camp, el Hamiltonià i de la secció transversal, respectivament.

Observació 6.2. Com s'ha vist a 4.1, per trobar solucions periòdiques, es pot utilitzar una combinació dels VEPs de la matriu diferencial del sistema. De manera que, com es demostrarà més endavant, es pot prendre com a condició inicial una corba parametritzada de la forma:

$$k(\theta) = L_i + r\big(\cos(2\pi\theta)\mathbf{u} - \sin(2\pi\theta)\mathbf{v}\big), \tag{6.6}$$

on $r \in \mathbb{R}$ és prou petit, $\mathbf{u}, \mathbf{v}$ són vectors que compleixen $\mathbf{w} = \mathbf{u} + \mathbf{iv}$, sent $\mathbf{w}$ un VEP de la matriu diferencial del sistema i L_i és un punt fix del camp $\dot{\mathbf{x}} = X(\mathbf{x})$ *i.e.* $X(L_i) = 0$. Al partir d'una aproximació, es tindrà que $G(k,\omega) = (E(\theta), e_h, e_g)^T$, i l'objectiu serà aproximar tant com es pugui la corba k perquè compleixi que $G(k,\omega) = 0$. L'aproximació partirà inicialment d'un error de l'ordre de r^2. Aquesta afirmació queda demostrada en (6.44).

6.2 Algorisme resum de resolució

Per aconseguir $G(k,\omega) = 0$, partint de (6.5), es considerarà una discretització d'una funció P-periòdica k a N punts de la grid, posant $\mathbb{K}_N := \{k_i\}_{i=0}^{N-1}$, on $k_i = k(\theta_i)$ i $\theta_i = \frac{P}{N} \cdot i$. Així, es pot definir una funció

$$G_N : \ \mathbb{R}^{m\cdot N} \times \mathbb{R} \longrightarrow \mathbb{R}^{m\cdot N} \times \mathbb{R}^2 \tag{6.7}$$

$$(\mathbb{K}_N, w) \longmapsto G_N(\mathbb{K}_N, \omega) = \begin{pmatrix} X_h(k(\theta_i)) - k'(\theta_i)\omega \\ h(k_0) - h_0 \\ g(k_0) - g_0 \end{pmatrix}. \tag{6.8}$$

L'objectiu és trobar les arrels d'aquesta funció. Partint de $G_N(\mathbb{K}_N, w) = (E_N, e_h, e_P)^T$, es farà un Newton modificat per a trobar uns elements que compleixin, menyspreant un ordre prou petit, $G_N(\mathbb{K}_N, w) = 0$. Conseqüentment, es consideraran unes correccions $\Delta\mathbb{K}$ i $\delta\omega$, de $\mathbb{K}$ i ω, respectivament. $\mathbb{K}$ i ω s'aniran actualitzant tal que

$$\begin{aligned} \mathbb{K} &\to \mathbb{K} + \Delta\mathbb{K} \\ \omega &\to \omega + \delta\omega \end{aligned} \tag{6.9}$$

Així, aplicant Taylor a primer ordre, s'obté

$$0 = G_N(\mathbb{K}_N + \Delta\mathbb{K}_N, \omega + \delta\omega) = G_N(\mathbb{K}_N, \omega) + D_{\mathbb{K}_N} G_N(\mathbb{K}_N, \omega)\Delta\mathbb{K}_N + \partial_\omega G_N(\mathbb{K}_N, \omega)\delta\omega + \mathcal{O}_2.$$

Menyspreant els valors d'ordre més gran o igual a dos, i aïllant G_N, s'obté

$$\begin{aligned} D_{\mathbb{K}_N} G_N(\mathbb{K}_N, \omega)\Delta\mathbb{K}_N + \partial_\omega G_N(\mathbb{K}_N, \omega)\delta\omega &= \begin{pmatrix} D_{\mathbb{K}_N} G_N(\mathbb{K}_N, \omega) \\ \partial_\omega G_N(\mathbb{K}_N, \omega) \end{pmatrix} \cdot \begin{pmatrix} \Delta\mathbb{K}_N \\ \delta\omega \end{pmatrix} \\ &= -G_N(\mathbb{K}_N, \omega) = \begin{pmatrix} -E_N \\ -e_h \\ -e_g \end{pmatrix}. \end{aligned} \tag{6.10}$$

Es busca la millor solució a través de solucionar aquest Newton modificat, seguint l'esquema de la Figura 7, on també s'hi aprecia un Newton amb continuació refinada.

Un cop s'ha calculat $(\mathbb{K}, \omega)$ per a un cert nivell d'energia h, posant

$$x(h)(\theta) = \begin{pmatrix} G_h(\theta) \\ \omega(\theta), \end{pmatrix} \tag{6.11}$$

es pot calcular $D_h G(x(h), h)$ fent servir el teorema de la funció implícita, que implica que

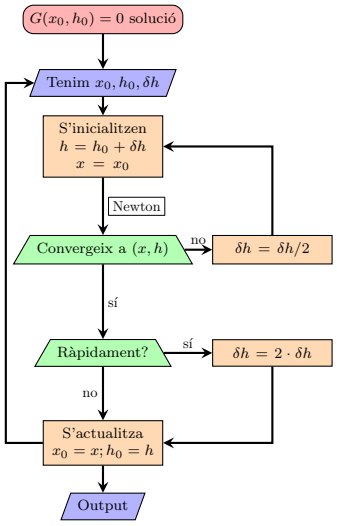
$$0 = D_{\mathbf{x}} G(\mathbf{x}(h), h) \cdot \mathbf{x}'(h) + \frac{\partial G}{\partial h}(\mathbf{x}(h), h), \tag{6.12}$$

d'on es calcula $\mathbf{x}'(h)$, necessari pel Newton refinat. Discretitzant aquest argument, s'obté que

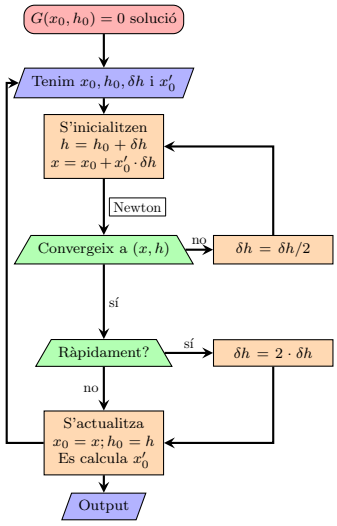
$$\begin{pmatrix} D_{\mathbb{K}_N} G_N(\mathbb{K}_N, \omega) \\ \delta_\omega G_N(\mathbb{K}_N, \omega) \end{pmatrix} \cdot \begin{pmatrix} \mathbb{K}'_N(h) \\ \omega'(h) \end{pmatrix} = -\frac{\partial G_N}{\partial h}(\mathbb{K}_N, \omega). \tag{6.13}$$

Notem que aquest sistema a resoldre és anàleg a (6.10). Així, la condició inicial de cada Newton passarà a ser l'aproximació més refinada:

$$x = x_0 + x'_0 \cdot \delta h. \tag{6.14}$$



(a) Mètode de Newton i continuació.



(b) Mètode de Newton i continuació refinada.

Figura 7: Diagrames de flux per als mètodes de Newton amb continuació.

6.3 Algorisme implementat detallat

Com s'ha vist a (6.2), per aconseguir l'òrbita periòdica, es vol aconseguir $X(k(\theta)) - k'(\theta)w = 0$, on $k : \mathbb{T} = \mathbb{R}/\mathbb{Z} \longrightarrow \mathbb{R}^m$.

Partirem d'una aproximació de l'òrbita, així doncs,

$$G(k,\omega) = \begin{pmatrix} X(k(\theta)) - k'(\theta)\omega \\ H(k(0)) - H_0 \\ g(k(0)) - g_0 \end{pmatrix} = \begin{pmatrix} E(\theta) \\ e_H \\ e_g \end{pmatrix}, \tag{6.15}$$

Per exemple, per **òrbites periòdiques planes**, es defineix g com l'aplicació lineal

$$\begin{aligned} g: \ & \mathbb{R}^m \longrightarrow \mathbb{R} \\ & \mathbf{x} \longmapsto g(\mathbf{x}) = \hat{\pi}_y \cdot \mathbf{x} = y, \end{aligned} \tag{6.16}$$

on $\hat{\pi}_y$ és la projecció en y, i $g_0 = 0$.

Nota. Per trobar les òrbites periòdiques verticals, substituiríem $\hat{\pi}_y$ per $\hat{\pi}_z$.

Considerant les correccions Δk i $\delta\omega$, de k i ω respectivament que minimitzen l'error, es fa la transformació:

$$k \to k + \Delta k \tag{6.17}$$

$$\omega \to \omega + \delta\omega \tag{6.18}$$

Desenvolupant i fent Taylor en $k(\theta)$,

$$\begin{aligned} 0 &= \overbrace{X\big(k(\theta) + \Delta k(\theta)\big)}^{X(k(\theta))+DX(k(\theta))\Delta k(\theta)+\mathcal{O}_2} + k'(\theta)\omega - \Delta k'(\theta)\omega - k'(\theta)\delta\omega - \delta k'(\theta)\delta\omega \\ &= E(\theta) + DX(k(\theta))\Delta k(\theta) - \Delta k'(\theta)\omega - k'(\theta)\delta w + \mathcal{O}_2 \end{aligned}$$

Menyspreant els errors d'ordre més gran o igual a 2, s'obté:

$$\boxed{DX(k(\theta))\Delta k(\theta) - \Delta k'(\theta)\omega - k'(\theta)\delta\omega = -E(\theta)}.$$

Anàlogament,

$$\begin{aligned} 0 = H(k(0)) + DH(k(0))\Delta k(0) - H_0 + \mathcal{O}_2 &\Rightarrow \boxed{DH(k(0))\Delta k(0) = -e_H}, \\ 0 = g(k(0)) + Dg(k(0))\Delta k(0) - g_0 + \mathcal{O}_2 &\Rightarrow \boxed{Dg(k(0))\Delta k(0) = -e_P}. \end{aligned}$$

Així, discretitzant la funció $k : \mathbb{T}^1 = \mathbb{R}/\mathbb{Z} \to \mathbb{R}^m$, en N punts, i notant $k_i = k(\theta_i)$ i $E_i = E(\theta_i)$, on $\theta(i) = \theta_i = \frac{i}{N}$, $i = 0 \div N-1$, s'obté:

$$D_{\mathbb{K}_N,\omega} G_N(\mathbb{K}_N,\omega) = \begin{pmatrix} DX(k_i)\Delta k_i - \Delta k_i'\omega - k_i'\delta\omega \\ DH(k_0)\Delta k_0 \\ g(\Delta k_0) \end{pmatrix} = \begin{pmatrix} -E_i \\ -e_h \\ -e_g \end{pmatrix} = -G_N(\mathbb{K}_N,\omega), \tag{6.19}$$

on $E_N \in \mathbb{R}^{Nm}$ és l'error del camp en els punts de la grid, e_H l'error en energia i e_g l'error de la secció transversal. En aquesta equació discretitzada a N punts de la grid, que

coincideix amb (6.10), es tenen $m \cdot N + 2$ equacions, i es vol que les $m \cdot N + 1$ incògnites siguin $(\{\Delta k_i\}_{i=0}^{N-1}, \delta w)$. Aparentment és un sistema sobredeterminat. Així doncs, intentem trobar la millor solució possible, on els errors siguin el més propers a zero possibles.

Per a tenir aquestes incògnites, es vol discretitzar un operador lineal tal que:

$$\Delta k \mapsto DX(k(\theta))\Delta k(\theta) - \Delta k'(\theta)\omega. \tag{6.20}$$

- L'operador discretitzat $\Delta k \mapsto DX(k(\theta))\Delta k(\theta)$ és trivial:

$$\begin{pmatrix} \Delta k_0 \\ \vdots \\ \Delta k_{N-1} \end{pmatrix}_{m\cdot N\times 1} \longmapsto \begin{pmatrix} DX(k_0) & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & DX(k_{N-1}) \end{pmatrix}_{m\cdot N\times m\cdot N} \begin{pmatrix} \Delta k_0 \\ \vdots \\ \Delta k_{N-1} \end{pmatrix}_{m\cdot N\times 1}$$

- Ara, es discretitza l'operador lineal per $\Delta k(\theta) \mapsto \Delta k'(\theta)$.

Reduïnt el problema a $m = 1$, s'ha de discretitzar l'operador derivada de la funció

$$\begin{aligned} D_1 \colon \mathcal{C}^\infty(\mathbb{T}, \mathbb{R}) &\longrightarrow \mathcal{C}^\infty(\mathbb{T}, \mathbb{R}) \\ \Delta k_1(\theta) &\mapsto \Delta k_1'(\theta). \end{aligned} \tag{6.21}$$

Es prenen com a base les funcions representades per $e_i = \{\delta_{ij}\}_{j,i=0\div N-1}$, on δ_{ij} és la **delta de Kronecker**[24]. Fixat un i, es busca un operador lineal $\hat{D}_1 \in \mathbb{R}^N \times \mathbb{R}^N$ que faci la transformació

$$\boxed{\{e_i(\theta_j) = \delta_{ij}\}_{j=0}^{N-1} \xrightarrow{\text{FFT}} e_i(\theta) = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} \hat{e}_{ik} e^{2\pi \mathbf{i} k\theta} \to e_i'(\theta) = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} (2\pi\mathbf{i}k)\hat{e}_{ik} e^{2\pi\mathbf{i}k\theta} \xrightarrow{\text{IFFT}} \{e_i'(\theta_j)\}_{j=0}^{N-1}.}$$

D'aquesta manera, s'obté la matriu, o operador discretitzat, $\hat{D}_1$ tal que

$$\Delta k_1(\theta) \mapsto \Delta k_1'(\theta) = \hat{D}_1 \Delta k_1(\theta). \tag{6.22}$$

Observació 6.3. Es pot trobar explícitament $\hat{e}_{ik}$. Vegem-ho:

Sigui $e_i(\theta) = \sum_{k=-N/2}^{N/2-1} \hat{e}_{ik} e^{\mathbf{i}2\pi k\theta}$, es vol obtenir explícitament la seva derivada. Per definició,

$$\hat{e}_{ik} = \frac{1}{N} \sum_{j=0}^{N-1} e_i\big(\frac{j}{N}\big) e^{-\mathbf{i}2\pi \frac{jk}{N}} = \frac{1}{N} \sum_{j=0}^{N-1} \delta_{ij} e^{-\mathbf{i}2\pi \frac{jk}{N}} = \frac{1}{N} e^{-\mathbf{i}2\pi \frac{ik}{N}}. \tag{6.23}$$

Aleshores,

$$e_i(\theta) = \frac{1}{N} \sum_{k=-N/2}^{N/2-1} e^{-\mathbf{i}2\pi \frac{ik}{N}} e^{\mathbf{i}2\pi k\theta}, \tag{6.24}$$

[24]La **delta de Kronecker** és una funció $\delta : \mathbb{Z} \times \mathbb{Z} \to \{0, 1\}$ definida com

$$\delta_{ij} = \delta_j^i = [i = j] = \begin{cases} 1, & \text{si } i = j \\ 0, & \text{si } i \neq j \end{cases}.$$

$$\text{on, per } k \in \{-\frac{N}{2}, \frac{N}{2}\}, \quad e^{-\mathbf{i}\pi i} = \begin{cases} 1, & \text{si } i \text{ parell} \\ 0, & \text{si } i \text{ senar} \end{cases}.$$

Així doncs,

$$e_i'(\theta) = \frac{1}{N} \sum_{k=-N/2}^{N/2-1} (\mathbf{i}2\pi k) e^{-\mathbf{i}2\pi \frac{ik}{N}} e^{\mathbf{i}2\pi k \frac{j}{N}}. \tag{6.25}$$

Per tant, existeix una fórmula explícita de D_1. □

Extrapolant el mètode per a qualsevol $m \in \mathbb{N}$ i prenent com a base la funció que en la grid val V_{ij}, on:

$$\begin{aligned} V_{ij} : \mathbb{T}^1 = \mathbb{R}/\mathbb{Z} &\longrightarrow \mathbb{R}^m \\ \theta_I &\longmapsto V_{ij}(\theta_I) = \begin{cases} 0 & \text{si } i \neq I \\ \delta_{jj} & \text{si } i = I \end{cases} \end{aligned} \tag{6.26}$$

En aquest cas, fixat un i, es busca un operador lineal $\hat{D}_m \in \mathbb{R}^{mN} \times \mathbb{R}^{mN}$ que faci la transformació

$$\boxed{\{V_{ij}(\theta_I)\}_{j=0}^{N-1} \to \{V_{ij}'(\theta_I)\}_{j=0}^{N-1}}.$$

Semblaria que s'ha de fer el procediment $m \times N$ vegades però, realment, només cal fer-lo N vegades, ja que les transformacions seran equivalents. És a dir, per a una mateixa θ, les m components de cadascuna de les funcions avaluades en els punts de la grid, seguiran la mateixa transformació. Vegem l'exemple per $m = 6$:

$$\begin{pmatrix} e_0(\theta) \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} e_0'(\theta) \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}; \quad \begin{pmatrix} 0 \\ e_0(\theta) \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} 0 \\ e_0'(\theta) \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}; \quad \dots \quad ; \quad \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ e_0(\theta) \end{pmatrix} \to \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ e_0'(\theta) \end{pmatrix}$$

Seguint el mateix procediment, es calcula $\hat{D}_1$. Un cop ja es té la matriu $\hat{D}_1$, es fa servir el producte de Kronecker[25] per obtenir la matriu desitjada, $\hat{D}_m = \hat{D}_1 \otimes I_m$:

[25] El **producte de Kronecker** entre dues matrius A i B, $A \otimes B$, és una matriu definida per:

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix} \; \textit{i.e.} \; (A \otimes B)_{(i-1)p+r,(j-1)q+s} = a_{ij}b_{rs},$$

on A és una matriu $m \times n$ amb elements a_{ij}, B és una matriu $p \times q$ amb elements b_{rs}, i la matriu resultant $A \otimes B$ té dimensions $mp \times nq$.

$$\hat{D}_m = \hat{D}_1 \otimes I_m = \begin{pmatrix} D_{00} & \cdots & D_{0,N-1} \\ \vdots & \ddots & \vdots \\ D_{N-1,0} & \cdots & D_{N-1,N-1} \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & \ddots & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & 1 \end{pmatrix} =$$

$$\left(\begin{array}{cccccc|c|c} D_{00} & 0 & 0 & 0 & 0 & 0 & & D_{0,N-1} \\ 0 & D_{00} & 0 & 0 & 0 & 0 & & \quad D_{0,N-1} \\ 0 & 0 & D_{00} & 0 & 0 & 0 & \cdots & \quad\quad D_{0,N-1} \\ 0 & 0 & 0 & D_{00} & 0 & 0 & & \quad\quad\quad D_{0,N-1} \\ 0 & 0 & 0 & 0 & D_{00} & 0 & & \quad\quad\quad\quad D_{0,N-1} \\ 0 & 0 & 0 & 0 & 0 & D_{00} & & \quad\quad\quad\quad\quad D_{0,N-1} \\ \hline \multicolumn{6}{c|}{\vdots} & \ddots & \multicolumn{2}{c}{\vdots} \\ \hline \multicolumn{6}{c|}{D_{N-1,0} \otimes I_m} & \cdots & \multicolumn{2}{c}{D_{N-1,N-1} \otimes I_m} \end{array}\right).$$

L'operador derivada discretitzat, doncs, serà

$$\hat{D}_m \cdot \begin{pmatrix} \Delta k_0 \\ \Delta k_1 \\ \vdots \\ \Delta k_{N-1} \end{pmatrix}_{(m\cdot N)\times 1} = \begin{pmatrix} \Delta k'_0 \\ \Delta k'_1 \\ \vdots \\ \Delta k'_{N-1} \end{pmatrix}_{(m\cdot N)\times 1} . \tag{6.27}$$

Observació 6.4. A l'hora de desenvolupar el codi, s'han fet diversos tests per a comprovar l'eficiència de l'operador derivada: comparant els resultats als que s'obtindrien via diferències finites i comparant els punts de k avaluats en la grid amb els de k', obtinguts mitjançant mètodes de Fourier.

Ja tenint l'operador discretitzat $\Delta k \mapsto DX(k(\theta))\Delta k(\theta) - \Delta k'(\theta)\omega$ i, havent calculat $k'(\theta_i)$, passant per Fourier, es pot reescriure (6.19) matricialment com

$$\left(\begin{array}{ccc|c} & & & -k'_0 \\ & DX(k(\theta_i)) - \hat{D}_m \cdot \omega & & -k'_1 \\ & & & \vdots \\ & & & -k'_{N-1} \\ \hline DH(k_0) & 0 & 0 & 0 \\ D_g(k_0) & 0 & 0 & 0 \end{array}\right)_{(N\cdot m+2)\times(m\cdot N+1)} \begin{pmatrix} \Delta k_0 \\ \Delta k_1 \\ \vdots \\ \Delta k_{N-1} \\ \hline \delta w \end{pmatrix}_{(m\cdot N+1)\times 1} = \begin{pmatrix} -E_0 \\ -E_1 \\ \vdots \\ -E_{N-1} \\ \hline -e_H \\ -e_P \end{pmatrix}_{(m\cdot N+2)\times 1} , \tag{6.28}$$

on $Dg = (\frac{\partial \pi}{\partial x}, \frac{\partial \pi}{\partial p_x}, \frac{\partial \pi}{\partial y}, ...) = (0, 0, 1, 0, 0, 0)$ i $DH = \left(\frac{\partial H}{\partial x}, \frac{\partial H}{\partial p_x}, \frac{\partial H}{\partial y}, \frac{\partial H}{\partial p_y}, \frac{\partial H}{\partial z}, \frac{\partial H}{\partial p_z}\right)$.

Per resoldre aquest sistema d'equacions lineals, s'escull el mètode de la SVD. La **descomposició de valor singular** (SVD) **compacta**[26] és una manera de representar una matriu A de dimensions $m \times n$ (on $m \geq n$) com el producte de tres matrius més simples:

$$A = U\Sigma V^T, \tag{6.29}$$

aquesta descomposició sempre és possible. On:

[26] La SVD habitual és amb una matriu U $m \times m$ i una Σ $m \times n$. S'usa la compacta per reaprofitar memòria.

- U és una matriu $m \times n$ ortogonal per columnes.
- Σ és una matriu simètrica $n \times n$ diagonal amb els valors singulars de A.
- V és una matriu ortogonal $n \times n$ que conté els vectors propis de $A^T A$.

Definició 6.5. *Els **valors singulars** d'una matriu A $m \times n$ són les arrels quadrades dels VAPs de $A^T A$ si $m \geq n$, i de AA^T si $m < n$. i.e. els valors singulars són $\sigma_i = \sqrt{\lambda_i}$, on λ_i els VAPs de $A^T A$ (o AA^T).*

Aquesta descomposició té moltes aplicacions pràctiques. Per exemple, en anàlisi de dades permet reduir la dimensionalitat de les dades mentre es conserva la informació més rellevant. En compressió d'imatges es pot utilitzar per emmagatzemar imatges de manera més eficient. En general, la SVD és una eina potent per entendre i processar matrius de dades en una àmplia gamma de problemes.

Per resoldre un sistema $Ax = b$ a través de la SVD, es parteix de la descomposició $A = U\Sigma V^T$ i s'obté que la solució amb norma d'error mínim seria $\bar{x} = A^\dagger b$, on $A^\dagger = V\Sigma^\dagger U^T$ és la pseudoinversa de la matriu A (*i.e.* $A^\dagger = (A^T A)^{-1} A^T$). Per tant,

$$\bar{x} = V \begin{pmatrix} \frac{1}{\sigma_1} & & \\ & \ddots & \\ & & \frac{1}{\sigma_n} \end{pmatrix} U^T b. \tag{6.30}$$

Observació 6.6. La descomposició SVD també és possible si $m < n$. En aquest cas, els valors singulars σ_j i les columnes de U seran zero per a tot $j > m$.

Un cop ja tenim el sistema resolt, aleshores s'actualitzen les condicions inicials per a

$$\begin{aligned} k_i &= k_i + \Delta k_i \\ \omega &= \omega + \delta\omega, \end{aligned} \tag{6.31}$$

es repeteix tot el procediment des de 6.15 fins que es consideri que el Newton ja ha convergit. *i.e.* fins que G o les correccions siguin prou petites.

Quan Newton hagi convergit, s'actualitzarà el nivell d'energia $h_0 = h + h_i$ per a determinar una nova òrbita de la família, utilitzant el triomf de l'òrbita anterior com a aproximació inicial d'aquesta nova òrbita.

6.4 Aplicació

El programa elaborat en aquest treball se centra en el sistema del RTBP i utilitza dues rutines del llibre de *Numerical Recipies in C*[9].

Es partirà del camp (3.27), aquest camp és Hamiltonià, així doncs, es podrà trobar una família d'òrbites periòdiques. En aquesta secció es buscarà una família d'òrbites periòdiques, tant planes com vestricals, al voltant dels punts col·lineals del sistema, seguint el Newton modificat que s'ha comentat a la secció prèvia 6.2.

La matriu diferencial del camp és la calculada en (3.40):

$$DX(L_i) = \left(\begin{array}{cccc|cc} 0 & 1 & 1 & 0 & 0 & 0 \\ -1-\partial_{xx}\bar{U} & 0 & -\partial_{xy}\bar{U} & 1 & -\partial_{xz}\bar{U} & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ -\partial_{yx}\bar{U} & -1 & -1-\partial_{yy}\bar{U} & 0 & -\partial_{yz}\bar{U} & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 1 \\ -\partial_{zx}\bar{U} & 0 & -\partial_{zy}\bar{U} & 0 & -\partial_{zz}\bar{U} & 0 \end{array}\right) = \left(\begin{array}{cc|c} \multicolumn{2}{c|}{\multirow{2}{*}{A_1}} & \\ & & \\ \hline & & \multirow{2}{*}{A_2} \\ & & \end{array}\right)$$

Per a calcular els VEPs, primer es calculen els VAPs de la matriu diferencial, que es notarà com $A := DX(L_i)$. Així doncs, recordant que els VAPs s'han calculat a la secció 3.5, s'obté la matriu de VAPs

$$\bar{\Lambda} = \begin{pmatrix} \lambda & 0 & 0 & 0 & 0 & 0 \\ 0 & -\lambda & 0 & 0 & 0 & 0 \\ 0 & 0 & \beta_p\mathbf{i} & 0 & 0 & 0 \\ 0 & 0 & 0 & -\beta_p\mathbf{i} & 0 & 0 \\ 0 & 0 & 0 & 0 & \beta_v\mathbf{i} & 0 \\ 0 & 0 & 0 & 0 & 0 & -\beta_v\mathbf{i} \end{pmatrix}, \tag{6.32}$$

on $\lambda = +\sqrt{-B+\sqrt{B^2-C}}$; $\beta_p = +\sqrt{B+\sqrt{B^2-C}}$, $\beta_v = \sqrt{\bar{U}_{zz}}$; i, es compleix

$$\begin{aligned} \mathbf{i}\beta_p &= \mathbf{i}\sqrt{B+\sqrt{B^2-C}} = \sqrt{-B-\sqrt{B^2-C}} \\ \mathbf{i}\beta_v &= \mathbf{i}\sqrt{\bar{U}_{zz}} = \sqrt{-\bar{U}_{zz}}. \end{aligned}$$

Partint de la matriu $\bar{\Lambda}$, es pot generar la matriu de VEPs P. Siguin $\mathbf{v}_1$, $\mathbf{v}_2$ els VEPs de λ, $-\lambda$, respectivament; $\mathbf{w}_p$, $\bar{\mathbf{w}}_p$ els VEPs de $\beta_p\mathbf{i}$, $-\beta_p\mathbf{i}$; i $\mathbf{w}_v$, $\bar{\mathbf{w}}_v\mathbf{i}$ els VEPs de $\beta_v\mathbf{i}$, $-\beta_v\mathbf{i}$:

$$DX(L_i)\cdot P = \overbrace{\begin{pmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{w}_p & \bar{\mathbf{w}}_p & \mathbf{w}_v & \bar{\mathbf{w}}_v \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}}^{P} \cdot \begin{pmatrix} \lambda & 0 & 0 & 0 & 0 & 0 \\ 0 & -\lambda & 0 & 0 & 0 & 0 \\ 0 & 0 & \beta_p\mathbf{i} & 0 & 0 & 0 \\ 0 & 0 & 0 & -\beta_p\mathbf{i} & 0 & 0 \\ 0 & 0 & 0 & 0 & \beta_v\mathbf{i} & 0 \\ 0 & 0 & 0 & 0 & 0 & -\beta_v\mathbf{i} \end{pmatrix}$$

Per treballar d'una manera més còmoda numèricament, es busca una matriu de VEPs alternativa que només contingui VEPs reals. Sigui $\mathbf{w}_p = \mathbf{u}_p + \mathbf{i}\mathbf{v}_p$,

$$\begin{cases} A(\mathbf{u}_p + \mathbf{i}\mathbf{v}_p) = & \mathbf{i}\beta_p(\mathbf{u}_p + \mathbf{i}\mathbf{v}_p) = \mathbf{i}\beta_p\mathbf{u}_p - \beta_p\mathbf{v}_p, \\ A(\mathbf{u}_p + \mathbf{i}\mathbf{v}_p) = & A\mathbf{u}_p + \mathbf{i}A\mathbf{v}_p \end{cases}. \tag{6.33}$$

Aleshores, igualant la part real i la part imaginària s'obté

$$\begin{cases} A\mathbf{u}_p = -\beta_p\mathbf{v}_p \\ A\mathbf{v}_p = \beta_p\mathbf{u}_p \end{cases} \tag{6.34}$$

Matricialment,

$$A \cdot \left(\mathbf{u}_p \mid \mathbf{v}_p\right) = \left(\mathbf{u}_p \mid \mathbf{v}_p\right) \cdot \left(\begin{array}{c|c} 0 & \beta_p \\ \hline -\beta_p & 0 \end{array}\right) \tag{6.35}$$

El procediment és anàleg per $\mathbf{w}_v = \mathbf{u}_v + \mathbf{i}\mathbf{v}_u$. Així doncs, notem $\hat{P} = (\mathbf{v}_1, \mathbf{v}_2, \mathbf{u}_p, \mathbf{v}_p, \mathbf{u}_v, \mathbf{v}_v)$ i, fent servir la següent igualtat:

$$A \cdot \hat{P} = \overbrace{\begin{pmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{u}_p & \mathbf{v}_p & \mathbf{u}_v & \mathbf{v}_v \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}}^{\hat{P}} \cdot \overbrace{\begin{pmatrix} \lambda & 0 & 0 & 0 & 0 & 0 \\ 0 & -\lambda & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \beta_p & 0 & 0 \\ 0 & 0 & -\beta_p & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \beta_v \\ 0 & 0 & 0 & 0 & -\beta_v & 0 \end{pmatrix}}^{\bar{\Lambda}}, \tag{6.36}$$

es poden calcular els VEPs reals aplicant, per exemple, el mètode de Gauss[27]. Per a calcular els imaginaris purs, es fa:

$$\begin{cases} A\mathbf{u}_p = -\beta_p \mathbf{v}_p \\ A\mathbf{v}_p = \beta_p \mathbf{u}_p \end{cases} \Leftrightarrow \begin{cases} A\mathbf{u}_p + \beta_p \mathbf{v}_p = 0 \\ A\mathbf{v}_p - \beta_p \mathbf{u}_p = 0 \end{cases} . \tag{6.37}$$

De fet, en general, per trobar VEPs complexos d'un VAP complex $a + b\mathbf{i}$ d'una matriu A, es fa

$$A(\mathbf{u} + \mathbf{i}\mathbf{v}) = A\mathbf{u} + \mathbf{i}A\mathbf{v} = (a + b\mathbf{i})(\mathbf{u} + \mathbf{i}\mathbf{v}) = (a\mathbf{u} - b\mathbf{v}) + \mathbf{i}(a\mathbf{v} + b\mathbf{u}), \tag{6.38}$$

i.e.

$$\begin{cases} A\mathbf{u} - a\mathbf{u} + b\mathbf{v} = 0 \\ A\mathbf{v} - a\mathbf{v} - b\mathbf{u} = 0 \end{cases} . \tag{6.39}$$

Matricialment,

$$\left(\begin{array}{c|c} A - a \cdot Id & b \cdot Id \\ \hline -b \cdot Id & A - a \cdot Id \end{array}\right) \left(\frac{\mathbf{u}}{\mathbf{v}}\right) = \bar{A} \left(\frac{\mathbf{u}}{\mathbf{v}}\right) = \left(\frac{\mathbf{0}}{\mathbf{0}}\right). \tag{6.40}$$

En el cas que ens ocupa, $a = 0$ i, si es volen calcular els imaginaris plans, $b = \beta_p$. Aleshores, el sistema a resoldre és:

$$\left(\begin{array}{c|c} A_1 & \beta_p \cdot Id \\ \hline -\beta_p \cdot Id & A_1 \end{array}\right) \left(\frac{\mathbf{u}}{\mathbf{v}}\right) = \bar{A} \left(\frac{\mathbf{u}}{\mathbf{v}}\right) = \left(\frac{0}{0}\right). \tag{6.41}$$

i.e.

$$\left(\begin{array}{ccc|ccc} a_{00} & \cdots & a_{0,m-1} & \beta_p & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{m-1,0} & \cdots & a_{m-1,m-1} & 0 & \cdots & \beta_p \\ \hline -\beta_p & \cdots & 0 & a_{00} & \cdots & a_{0,m-1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & -\beta_p & a_{m-1,0} & \cdots & a_{m-1,m-1} \end{array}\right) \left(\begin{array}{c} u_0 \\ \vdots \\ u_{m-1} \\ \hline v_0 \\ \vdots \\ v_{m-1} \end{array}\right) = \left(\begin{array}{c} 0 \\ \vdots \\ 0 \\ \hline 0 \\ \vdots \\ 0 \end{array}\right). \tag{6.42}$$

[27] Johann Carl Friedrich Gauss (1777-1855): matemàtic, astrònom, geodesista i físic alemany.

Reordenem per poder resoldre amb Gauss i poder substituir per 1 i per 0 els últims elements d'ambdós VEPs: el real i l'imaginari, igualant-los a 1 i a 0, respectivament ($u_{m-1} = 1$, $v_{m-1} = 0$), triangular la matriu i anar solucionant el sistema substituint amunt:

$$\begin{pmatrix} \bar{a}_{00} & \bar{a}_{0,m} & \bar{a}_{0,1} & \cdots & \bar{a}_{0,m-1} & \bar{a}_{0,2m-1} \\ \bar{a}_{10} & \bar{a}_{1,m} & \bar{a}_{1,1} & \cdots & \bar{a}_{1,m-1} & \bar{a}_{1,2m-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \bar{a}_{m-1,0} & \bar{a}_{m-1,m} & \bar{a}_{m-1,1} & \cdots & \bar{a}_{m-1,m-1} & \bar{a}_{m-1,2m-1} \\ \bar{a}_{m,0} & \bar{a}_{m,m} & \bar{a}_{m,1} & \cdots & \bar{a}_{m,m-1} & \bar{a}_{m,2m-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \bar{a}_{2m-1,0} & \bar{a}_{2m-1,m} & \bar{a}_{2m-1,1} & \cdots & \bar{a}_{2m-1,m-1} & \bar{a}_{2m-1,2m-1} \end{pmatrix} \begin{pmatrix} u_0 \\ v_0 \\ u_1 \\ \vdots \\ v_{m-2} \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Fent això, s'obté $\mathbf{u}_p$ i $\mathbf{v}_p$.

Ara que ja s'ha fet aquest estudi respecte els VAPs i els VEPs de la matriu, la primera aproximació de l'òrbita que es prendrà, per a calcular una família d'òrbites periòdiques al voltant dels diferents punts col·lineals, serà

$$\begin{aligned} k\colon \mathbb{R} &\to \mathbb{R}^m \\ \theta &\mapsto k(\theta) = L_i + (\mathbf{u}\cos(2\pi\theta) - \mathbf{v}\sin(2\pi\theta))r, \end{aligned} \tag{6.43}$$

on $\mathbf{u}$ i $\mathbf{v}$ seran $\mathbf{u}_p$ i $\mathbf{v}_p$ per a calcular òrbites periòdiques planes, i seran $\mathbf{u}_v$ i $\mathbf{v}_v$ per a calcular òrbites periòdiques verticals. Prenent com a freqüència $\omega = \frac{\beta}{2\pi} > 0$, vegem, fent Taylor sobre L_i, que $X(K(\theta)) - k'(\theta)\omega$ és zero menyspreant un error de segon ordre:

$$\begin{aligned} & X(k(\theta)) - k'(\theta)\omega \\ &= X\big(L_i + (\mathbf{u}\cos(2\pi\theta) - \mathbf{v}\sin(2\pi\theta))r\big) - \mathbf{u}(-\sin(2\pi\theta))2\pi r\omega + \mathbf{v}\cos(2\pi\theta)2\pi r\omega \\ &= X(L_i) + DX(L_i)[r\mathbf{u}\cos(2\pi\theta) - r\mathbf{u}\sin(2\pi\theta)] + \mathcal{O}(r^2) + \mathbf{u}\sin(2\pi\theta)r\beta + \mathbf{v}\cos(2\pi\theta)r\beta \\ &= r(-\beta\mathbf{v})\cos(2\pi\theta) - r\beta\mathbf{u}\sin(2\pi\theta) + \mathcal{O}(r^2) + r\beta\mathbf{u}\sin(2\pi\theta) + r\beta\mathbf{v}\cos(2\pi\theta) \\ &= \mathcal{O}(r^2). \end{aligned} \tag{6.44}$$

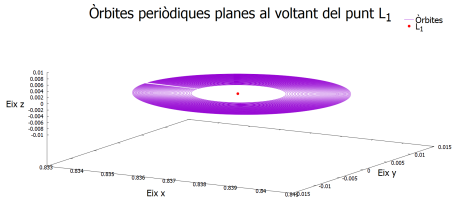
6.5 Resultats

Les famílies d'òrbites periòdiques al voltant dels punts de Lagrange L_1, L_2 i L_3 en el RTBP il·lustren com les trajectòries dels cossos en aquests punts són determinades pels nivells d'energia específics. Aquestes òrbites, obtingudes amb simulacions numèriques a través de l'algorisme i el codi desenvolupat, ofereixen una visió clara de la dinàmica que es desenvolupa al voltant dels punts de Lagrange.

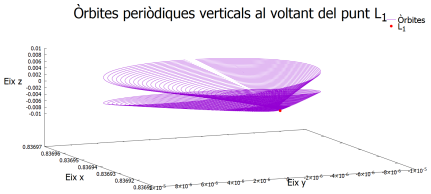
Considerant el RTBP del sistema Terra-Lluna, on

$$\mu = 1.215058560962404^{[7]},$$

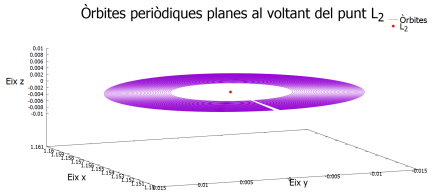
les gràfiques que s'obtenen són les de la Figura 8.



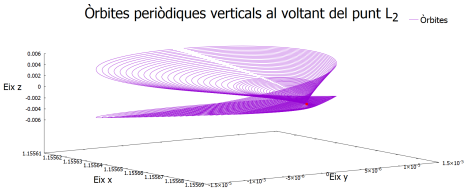
(a) Família d'òrbites periòdiques planes al voltant del punt L_1.



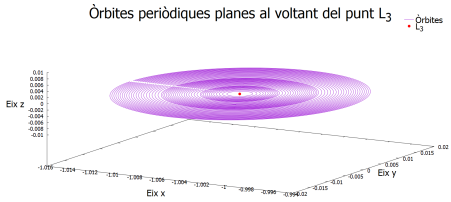
(b) Família d'òrbites periòdiques verticals al voltant del punt L_1.



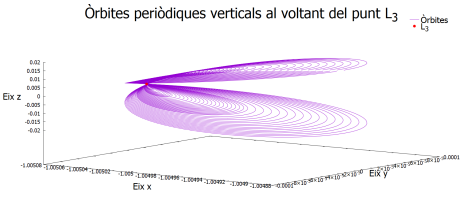
(c) Família d'òrbites periòdiques planes al voltant del punt L_2.



(d) Família d'òrbites periòdiques verticals al voltant del punt L_2.



(e) Família d'òrbites periòdiques planes al voltant del punt L_3.



(f) Família d'òrbites periòdiques verticals al voltant del punt L_3.

Figura 8: Famílies d'òrbites periòdiques al voltant dels punts de Lagrange.

Cada òrbita representada en les gràfiques de la Figura 8 correspon a un nivell d'energia. Les òrbites més properes als punts de Lagrange (L_1, L_2 i L_3) són les associades a nivells d'energia més baixos, mentre que les òrbites que s'allunyen progressivament d'aquests punts corresponen a nivells d'energia més alts.

Les òrbites verticals (graficades a la columna de la dreta de la Figura 8) mostren una distribució tridimensional, on la variació en l'eix z és notable; mentre que les òrbites planes (graficades a la columna de l'esquerra de la Figura 8) es mantenen principalment en el pla horitzontal on $z = 0$. Aquesta diferència entre òrbites verticals i planes mostra com els diferents nivells d'energia influeixen en les dimensions i la complexitat de les òrbites.

Les òrbites verticals al voltant dels punts L_1, L_2 i L_3 segueixen patrons espiralats, cosa que indica una dinàmica complexa però periòdica: com més elevada és l'energia, més grans i més complicades són les òrbites. D'altra banda, les òrbites planes al voltant d'aquests punts es mantenen al pla horitzontal: les variacions en l'energia afecten principalment l'amplitud i la forma de les òrbites dins del mateix pla.

Les diferències entre les òrbites al voltant de L_1, L_2 i L_3 poden ser subtils però importants. Així doncs, per a aplicacions concretes com la planificació de missions espacials, es necessita un anàlisi detallat en la localització i l'estabilitat orbital.

Per tant, les òrbites periòdiques al voltant dels punts de Lagrange L_1, L_2 i L_3 mostren una clara dependència dels nivells d'energia, amb òrbites més petites i properes associades a energies més baixes i òrbites més grans i complexes associades a energies més altes. Aquesta comprensió de la relació entre energia i configuració orbital és essencial per a l'estudi de la dinàmica dels tres cossos i té importants implicacions pràctiques en astrofísica i exploració espacial.

Efectivament, una manera convenient de representar una família d'òrbites periòdiques obtinguda per continuació numèrica és fent un gràfic del període (o la freqüència) de les òrbites respecte la seva energia[7]. La figura 9 representa la freqüència de la família de Lyapunov planar de L_3 en el sistema Terra-Lluna del RTBP en funció de l'energia.

Definició 6.7. *La corba que representa el període (o la freqüència) respecte l'energia s'anomena* ***corba característica****.*

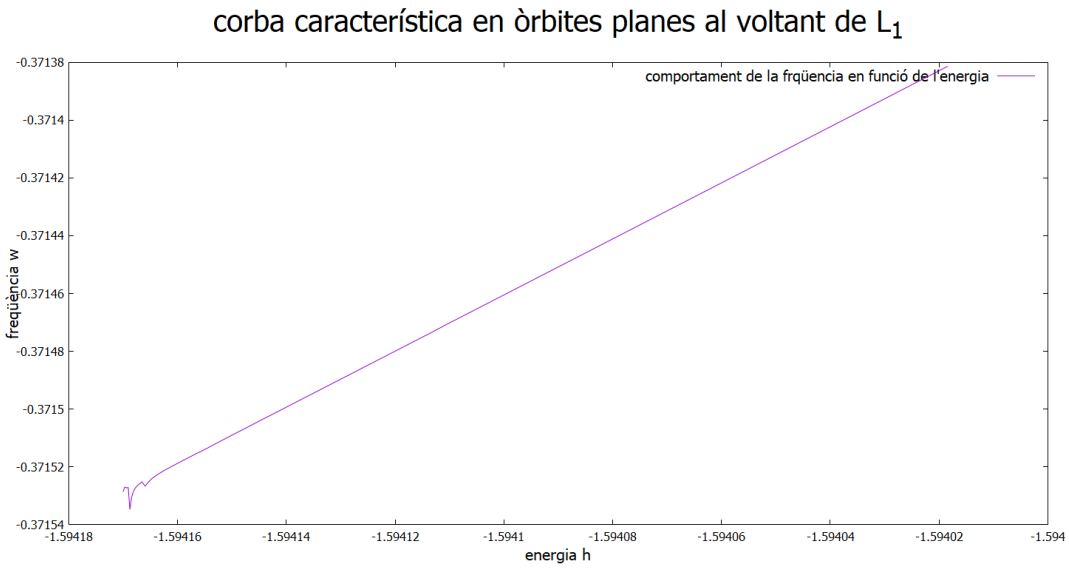


Figura 9: Relació entre energia i freqüència en les òrbites planes al voltant del punt L_1.

La Figura 9 mostra la relació entre energia i freqüència ω en les òrbites planes al voltant del punt L_1 (passa similarment en L_2 i L_3). Il·lustra com la freqüència de les òrbites varia en funció de l'energia. En aquest cas, s'observa una variació en la freqüència a mesura que l'energia canvia, ω sembla tenir un comportament gairebé lineal respecte a l'energia, la qual indica una dependència directa i més predictible de la freqüència orbital en funció de l'energia. Aquesta linealitat pot ser útil per a prediccions més precises en la dinàmica orbital d'aquests punts.

7 Conclusions

En aquest treball de final de grau, s'ha abordat el problema restringit dels tres cossos i l'estudi de les òrbites periòdiques associades a aquest, aplicant tècniques de la mecànica clàssica i de l'anàlisi de Fourier. S'ha desenvolupat un algorisme capaç de calcular aquestes òrbites, basant-se en les transformades de Fourier i aplicant-lo a sistemes Hamiltonians.

L'estudi realitzat estima el càlcul de la trajectòria que seguiria un tercer cos si es col·loqués en les proximitats de les famílies d'òrbites periòdiques identificades. Aquestes trajectòries proporcionen una visió teòrica de les possibles dinàmiques del sistema, contribuint al nostre coneixement sobre la naturalesa de les interaccions gravitatòries en un context idealitzat.

Tanmateix, cal tenir en compte que el món físic presenta una complexitat major. La realitat incorpora nombrosos factors addicionals que no es poden capturar completament en un model matemàtic simplificat, com les perturbacions externes, efectes relativistes, i altres influències que poden desviar les òrbites de la periodicitat prevista. Per tant, s'ha de tenir en compte que les solucions obtingudes en aquest estudi són ideals i serveixen com a aproximacions útils per entendre la dinàmica del sistema, però no reflecteixen completament la complexitat del moviment dels cossos celestes en l'espai real.

En resum, aquest treball ha proporcionat una metodologia rigorosa per a l'estudi d'òrbites periòdiques en el problema restringit dels tres cossos i ha posat de manifest la importància d'aplicar tècniques avançades d'anàlisi matemàtica i computacional per abordar problemes de dinàmica celeste. Tot i les limitacions inherents al model utilitzat, els resultats obtinguts ofereixen una base sòlida per a futurs estudis i aplicacions en astrodinàmica i altres camps relacionats.

Referències

[1] Arnold, V. I. (1989). *Mathematical Methods of Classical Mechanics* (2nd ed.). Springer International Publishing AG.

[2] Canadell, M.; Figueras, J.-L.; Haro, A.; Luque, A.; Mondelo, J.-M. (2015). *The Parameterization Method for Invariant Manifolds*. Applied Mathematical Sciences. Springer.

[3] Carroll, S. (2022). *The Biggest Ideas in the Universe: Space, Time and Motion*. Dutton.

[4] Cooley, J. W.; Tukey, J. W. (1965). *An Algorithm for the Machine Calculation of Complex Fourier Series*. `https://web.stanford.edu/class/cme324/classics/cooley-tukey.pdf`

[5] Crater, H. W. (1978). *Generalization of the Lagrange Equilateral-Triangle Solution and the Euler Collinear Solution to Nongravitational Forces in the Three-Body Problem*. Physical Review D, 17(4), 976. `https://doi.org/10.1103/PhysRevD.17.976`

[6] Koon, W. S.; Lo; M. W.; Marsden, J. E.; Ross, S. D. (2022). *Dynamical Systems, the Three-Body Problem, and Space Mission Design* (3rd ed.).

[7] Mondelo, J.-M. (2019). *Satellite Dynamics and Space Missions: Computing Invariant Manifolds for Libration Point Missions*. Springer INdAM Series, 34, 159-222.

[8] Pollard, H. (1966). *Mathematical Introduction to Celestial Mechanics*. Englewood Cliffs, New Jersey: Prentice-Hall.

[9] Press, W. H.; Teukolsky, S. A.; Vetterling, W. T.; Flannery, B. P. (1992). *Numerical Recipes in C: The Art of Scientific Computing* (2nd ed.). Cambridge University Press.

[10] Schmidt, D. S. (1978). *Hopf's Bifurcation Theorem and the Center Theorem of Liapunov with Resonance Cases*. Journal of Mathematical Analysis and Applications, 63, 354-370.

[11] Walker, J. S. (1996). *Fast Fourier Transforms* (2nd ed.). Boca Raton, FL: CRC Press.

[12] Wolfram, S. (2002). *A New Kind of Science*. Champaign, IL: Wolfram Media.

ANNEXOS

A Teoremes

A.1 Fórmula d'Euler Lagrange a partir del Principi de Hamilton

Recordem que el Principi de Hamilton diu que: "*El moviment del sistema a l'espai de configuració és tal que l'acció* $S = \int_{t_0}^{t_1} L(\mathbf{q}, \dot{\mathbf{q}}, t)dt$*, que és una integral de línia a l'espai de configuració, té un valor estacionari*".

Per entendre bé el que ens diu aquest principi, convé tenir en ment: L, que és una quantitat concreta que anomenem **lagrangià**; V, el **potencial generalitzat** (*i.e.* l'energia potencial que pot dependre també de les velocitats i el temps); i el concepte de **valor estacionari**.

Definició A.1. *Es diu que una funció té un* ***valor estacionari*** *si té el mateix valor canviant infinitessinalment el camí respecte l'original (menyspreant els infinitèssims a partir de segon ordre).*

Si α és un infinitèssim, fent una expansió de Taylor en $\alpha = 0$:

$$S(\alpha) = S(0) + \frac{dS}{d\alpha}_{\alpha=0} \alpha + \mathcal{O}(\alpha^2) \tag{A.1}$$

I la condició de que sigui un *valor estacionari*, negligint els termes quadràtics en α:

$$S(\alpha) - S(0) = \left.\frac{dS}{d\alpha}\right|_{\alpha=0} \alpha = 0 \Rightarrow \left.\frac{dS}{d\alpha}\right|_{\alpha=0} = 0 \tag{A.2}$$

Definició A.2. *Un* ***desplaçament virtual*** *és un canvi infinitesimal de les coordenades d'un sistema, compatible amb els seus lligams i suposadament realitzat de manera instantània en un moment determinat.*

Si posem $\left.\frac{d}{d\alpha}\right|_{\alpha=0} d\alpha := \delta$, podem escriure el Principi de Hamilton com:

$$\delta S = \delta \int_{t_0}^{t_1} L(\mathbf{q}, \dot{\mathbf{q}}, t)dt = 0. \tag{A.3}$$

Vegem com, del Principi de Hamilton, en podem extreure directament les equacions d'Euler-Lagrange:

Considerem el Lagrangià $L(\mathbf{q}, \dot{\mathbf{q}}, t)$, on $\mathbf{q} = (q_1, ..., q_n)$ té n coordenades diferents amb trajectòries $\mathbf{q}(t)$ i $\dot{\mathbf{q}}(t)$, per $i = 1 \div n$. Volem trobar una trajectòria tal que $S = \int_{t_0}^{t_1} L(\mathbf{q}, \dot{\mathbf{q}}, t)dt$ tingui un valor estacionàri. Per a això, definim n camins variats $\mathbf{q}(t) = \mathbf{q}(t, 0) + \alpha\eta(t)$, on $\mathbf{q}, \eta \in \mathcal{C}^3(\mathbb{R})$ i $\eta(t_0) = \eta(t_1) = 0$. Tenint així una trajectòria infinitessimalment variada que comença i acaba al mateix lloc. Reescrivint la condició de valor estacionari (A.2):

$$\frac{d}{d\alpha} \int_{t_0}^{t_1} L(\mathbf{q}(\alpha), \dot{\mathbf{q}}(\alpha), t)dt \Big|_{\alpha=0} d\alpha = 0 \tag{A.4}$$

Introduïnt la derivació en l'integrand obtenim:

$$\int_{t_0}^{t_1} \sum_i^n \Big(\frac{\partial L}{\partial q_i} \frac{\partial q_i}{\partial \alpha} + \frac{\partial L}{\partial \dot{q}_i} \frac{\partial \dot{q}_i}{\partial \alpha} \Big) dt \Big|_{\alpha=0} d\alpha = 0 \tag{A.5}$$

Integrant el segon integrand per parts, obtenim:

$$\int_{t_0}^{t_1} \sum_i^n \frac{\partial L}{\partial q_i} \frac{\partial \dot{q}_i}{\partial \alpha} dt = \sum_i^n \left(\frac{\partial L}{\partial \dot{q}_i} \frac{\partial q_i}{\partial \alpha} \Big|_{t_0}^{t_1} - \int_{t_0}^{t_1} \frac{\partial q_i}{\partial \alpha} \frac{d}{dt} \frac{\partial L}{\partial \dot{q}_i} dt \right) \tag{A.6}$$

El primer terme s'anul·la perquè $\frac{\partial q_i}{\partial \alpha} = \eta_i(t)$ i $\eta_i(t_1) = \eta_i(t_0) = 0$. Així:

$$\int_{t_0}^{t_1} \sum_i \left(\frac{\partial L}{\partial q_i} - \frac{d}{dt} \frac{\partial L}{\partial \dot{q}_i} \right) \eta(t) dt \Big|_{\alpha=0} d\alpha = 0 \tag{A.7}$$

Observant que $\frac{\partial q_i}{\partial q_i}\big|_{\alpha=0} d\alpha = \delta q_i$ *i.e.* representen els desplaçaments virtuals en cadascuna de les coordenades, podem posar:

$$\int_{t_0}^{t_1} \sum_i \left[\left(\frac{\partial L}{\partial q_i} - \frac{d}{dt} \frac{\partial L}{\partial \dot{q}_i} \right) \delta q_i \right] dt = 0 \tag{A.8}$$

I com que les coordenades són independents, cadascun dels integrands que es sumen ha de ser nul. Així, fent servir el lema 2.6, obtenim que

$$\frac{\partial L}{\partial q_i} - \frac{d}{dt} \frac{\partial L}{\partial \dot{q}_i} = 0 \tag{A.9}$$

que corresponen a les equacions d'Euler-Lagrange. □

A.2 Teorema de Lyapunov

Els resultats d'aquesta demostració estan basats, principalment, en l'article [10].

Definició A.3. *Sigui $\dot{\mathbf{x}} = f(\mathbf{x})$ un sistema d'equacions diferencials i sigui $\Omega \subset \mathbb{R}^n$ l'espai de fase*[28]*, es considera una funció $g : \Omega \to \mathbb{R}$ no localment constant amb derivades contínues en Ω. Es diu que $g(\mathbf{x})$ és **integral primera de** $\dot{\mathbf{x}} = f(\mathbf{x})$ si, per a tot $\mathbf{x}$,*

$$Dg(\mathbf{x})f(\mathbf{x}) = 0, \tag{A.10}$$

fet que implica que g és constant sobre les solucions del sistema.

Teorema A.4 (Lyapunov)**.** *Considerem que el sistema n-dimensional autònom*

$$\dot{\mathbf{x}} = A\mathbf{x} + f(\mathbf{x}), \tag{A.11}$$

on f és una funció diferenciable tal que, en $\mathbf{x} = \mathbf{0}$, valen zero ella i les seves primeres derivades. Suposem que el sistema admet una integral primera de la forma $I(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T S\mathbf{x} + o(|\mathbf{x}|^2)$, amb $S = S^T$ i $det(S) \neq 0$. Suposem també que el sistema té un l'origen un punt d'equacions amb exponents $\pm\mathbf{i}\beta, \lambda_1, ..., \lambda_n$ on $\mathbf{i}\beta$ és un imaginari pur. Si $\frac{\lambda_j}{\mathbf{i}\beta} \notin \mathbb{Z}$, per $j = 3, ..., n$, aleshores existeix una família uniparamètrica d'òrbites periòdiques que quan tendeixen al punt d'equilibri el seu període tendeix a $T = \frac{2\pi}{\beta}$.

Per tal de demostrar aquest Teorema, utilitzarem el teorema de la bifurcació de Hopf [10], que és un teorema més generalitzat d'aquest; i veurem que les hipòtesis del teorema de Lyapunov verifiquen les del teorema de la Bifurcació de Hopf.

Teorema A.5 (de la Bifurcació de Hopf)**.** *Considerem un sistema autònom n-dimensional d'equacions diferencials ordinàries donat per*

$$\dot{\mathbf{x}} = F(\mathbf{x}, \mu), \tag{A.12}$$

que depèn d'un paràmetre real μ. Sigui $F(\mathbf{x}, \mu)$ de classe $\mathcal{C}^2$ en les dues variables i $\mathbf{x} = \mathbf{x}(\mu)$ una família analítica de punts d'equilibri tal que $F(\mathbf{x}(\mu), \mu) = 0$. Sense pèrdua de generalitat, assumim que aquesta família ve donada per $\mathbf{x} \equiv \mathbf{0}$ i.e. $F(\mathbf{0}, \mu) = 0$. Suposem que per un cert valor de μ, posem $\mu = 0$, la matriu $F_{\mathbf{x}}(\mathbf{0}, \mu)$ té dos VAPs imaginaris purs $\pm\mathbf{i}\beta$, i cap altre enter múltiple de $\mathbf{i}\beta$ com a VAP. Si $\alpha(\mu) + \mathbf{i}\beta(\mu)$ és la continuació del VAP $\mathbf{i}\beta$ (per tant $\alpha(0) = 0$), assumim que $\alpha'(0) \neq 0$.

Aleshores, tenim que existeixen funcions diferenciables $\mu = \mu(\varepsilon)$ i $T = T(\varepsilon)$ depenen d'un paràmetre ε amb $\mu(0) = 0$ i $T(0) = 2\pi\beta^{-1}$ tals que hi ha solucions periòdiques no constants $x(t, \varepsilon)$ de (A.12) amb període $T(\varepsilon)$ tendint a l'origen quan $\varepsilon \to 0$.

Demostració. Fent el desenvolupament de Taylor de $\dot{\mathbf{x}} = F(\mathbf{x}, \mu)$ en $\mathbf{x} = \mathbf{0}$, s'obté:

$$\begin{aligned} \dot{\mathbf{x}} = \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{y} \end{pmatrix} &= F(\mathbf{x}, \mu) = F(0, \mu) + D_{\mathbf{x}}F(\mathbf{0}, \mu)\mathbf{x} + \mathcal{O}(\mathbf{x}^2) \\ &= \begin{pmatrix} \alpha(\mu) + \mathbf{i}\beta(\mu) & & \\ & \alpha(\mu) - \mathbf{i}\beta(\mu) & \\ & & B(\mu) \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ y \end{pmatrix} + \mathcal{O}(\mathbf{x}^2) \end{aligned} \tag{A.13}$$

[28] L'**espai de fase** és un conjunt multidimensional en el qual cada dimensió representa una variable que descriu l'estat del sistema en un instant donat. Els punts en aquest espai corresponen als possibles estats del sistema.

Fent un canvi de coordenades i, posant $\tau = \beta(\mu)t$, obtenim $\frac{d\tau}{dt} = \beta(\mu)$ i $\frac{d}{d\tau} = \frac{1}{\beta(\mu)}\frac{d}{dt}$. Així, fent servir la següent notació:

$$t\tfrac{d}{dt} :=^{\cdot} \quad ; \qquad \tau\tfrac{d}{d\tau} :=' \quad ,$$

s'obté:

$$\dot{x}_1 = t\frac{d}{dt}x_1 = t\beta(\mu)\frac{d}{d\tau}x_1 = \frac{t}{t}\tau\frac{d}{d\tau}x_1' = (\alpha(\mu) + \mathbf{i}\beta(\mu))x_1 + \hat{f}_1(x_1, x_2, y, \mu)$$
$$\dot{x}_2 = t\frac{d}{dt}x_2 = t\beta(\mu)\frac{d}{d\tau}x_2 = \frac{t}{t}\tau\frac{d}{d\tau}x_2' = (\alpha(\mu) - \mathbf{i}\beta(\mu))x_2 + \hat{f}_2(x_1, x_2, y, \mu)$$

Fent un canvi d'escala posant $a(\mu) = \frac{\alpha(\mu)}{\beta(\mu)}$, obtenim l'equació (A.12) com:

$$\begin{aligned} \dot{x}_1 &= (a(\mu) + \mathbf{i})x_1 + f_1(x_1, x_2, y, \mu) \\ \dot{x}_2 &= (a(\mu) - \mathbf{i})x_2 + f_2(x_1, x_2, y, \mu), \\ \dot{y} &= B(\mu)y + g(x_1, x_2, y, \mu) \end{aligned} \tag{A.14}$$

on $y \in \mathbb{R}^{n-2}$ és un vector, i $B(\mu) = B_0 + \mu(0)$ una matriu quadrada $(n-2)\times(n-2)$ no necessàriament en forma canònica i, per hipòtesi, B_0 no té cap VAP de la forma $n \cdot \mathbf{i}$, on $n \in \mathbb{Z}$. Les solucions reals venen donades només quan $x_1 = \bar{x}_1$, per tant es compleix que $f_1(x_1, \bar{x}_2, y, \mu) = \bar{f}_2(x_2, x_1, y, \mu)$. A més a més, les funcions f_1, f_2, g i les seves primeres derivades parcials s'anul·len si $x_1 = x_2 = y = 0$.

Introduïm un factor d'escala ϵ al sistema A.14, tal que $\mu = \epsilon\mu_1$, $x_j = \epsilon^2\xi_j$, on $j = 1, 2$, i $y = \epsilon^2\eta$ i, obtenim,

$$\begin{aligned} \epsilon^2\dot{\xi}_1 &= (a(\epsilon\mu_1) + \mathbf{i})\epsilon^2\xi_1 + f_1(\epsilon^2\xi_1, \epsilon^2\xi_2, \epsilon^2\eta, \epsilon\mu_1) \\ \epsilon^2\dot{\xi}_2 &= (a(\epsilon\mu_1) - \mathbf{i})\epsilon^2\xi_2 + f_2(\epsilon^2\xi_1, \epsilon^2\xi_2, \epsilon^2\eta, \epsilon\mu_1). \\ \epsilon^2\dot{\eta} &= B(\epsilon\mu_1)\epsilon^2\eta + g(\epsilon^2\xi_1, \epsilon^2\xi_2, \epsilon^2\eta, \epsilon\mu_1) \end{aligned} \tag{A.15}$$

Recordant que $a(0) = 0$ i considerant el desenvolupament de Taylor $a(\epsilon\mu_1) = a(0) + \epsilon\mu_1 a'(0) + \frac{(\epsilon\mu_1)^2}{2}a''(0) + \mathcal{O}((\epsilon\mu)^3) = \epsilon\mu_1 a'(0) + \mathcal{O}((\epsilon\mu))^2)$, s'obté:

$$\begin{aligned} \epsilon^2\dot{\xi}_1 &= (\epsilon\mu_1 a'(0) + \mathbf{i})\epsilon^2\xi_1 + f_1(\epsilon^2\xi_1, \epsilon^2\xi_2, \epsilon^2\eta, \epsilon\mu_1) \\ \epsilon^2\dot{\xi}_2 &= (\epsilon\mu_1 a'(0) - \mathbf{i})\epsilon^2\xi_2 + f_2(\epsilon^2\xi_1, \epsilon^2\xi_2, \epsilon^2\eta, \epsilon\mu_1). \\ \epsilon^2\dot{\eta} &= B(\epsilon\mu_1)\eta + g(\epsilon^2\xi_1, \epsilon^2\xi_2, \epsilon^2\eta, \epsilon\mu_1) \end{aligned} \tag{A.16}$$

Dividint a banda i banda per ϵ^2:

$$\begin{aligned} \dot{\xi}_1 &= \mathbf{i}\xi_1 + \epsilon\mu_1 a'(0)\xi_1 + \mathcal{O}(\epsilon^2) \\ \dot{\xi}_2 &= -\mathbf{i}\xi_2 + \epsilon\mu_1 a'(0)\xi_2 + \mathcal{O}(\epsilon^2), \\ \dot{\eta} &= B_0\eta + \mathcal{O}(\epsilon^2) \end{aligned} \tag{A.17}$$

En la cerca de solucions periòdiques de període proper a 2π, veiem que:

- Per $\epsilon = 0$, hem de tenir $\eta = 0$ mentre que ξ_1 i ξ_2 poden tenir condicions inicials arbitràries. Pel caràcter autònom del sistema i pel nostre factor d'escala, podem prendre com a condició incial: $\xi_1(0) = \xi_2(0) = 1$ i, per tant, obtenir la solució 2π-periòdica $\xi_1 = e^{\mathbf{i}t}$, $\xi_2 = e^{-\mathbf{i}t}$, $\eta = 0$.

- Per $\epsilon \neq 0$, podem esperar trobar solucions quasi periòdiques de període $T = 2\pi(1 - \epsilon\delta)$ si es satisfà la condició de periodicitat

$$\xi_j(T) - \xi_j(0) = 0 \quad j = 1, 2$$

$$\eta(T) - \eta(0) = 0.$$

Amb $\xi_1(0) = \xi_2(0) = 1$ i $\eta(0) = \eta_0$, tenim que les solucions de (A.17) esdevenen de la forma:

$$(1) : \eta(t) = \eta_0 e^{B_0 t} + \mathcal{O}(\epsilon^2) \Rightarrow \eta(T) = \eta_0 e^{2\pi B_0 + 2\pi\epsilon\delta B_0} + \mathcal{O}(\epsilon^2)$$

$$(2) : \xi_1(t) = e^{(\mathbf{i}+\epsilon\mu_1 a'(0))t} + \mathcal{O}(\epsilon^2) \Rightarrow$$

$$\xi_1(T) = e^{(\mathbf{i}+\epsilon\mu_1 a'(0))2\pi(1-\epsilon\delta)} + \mathcal{O}(\epsilon^2) = e^{2\pi\mathbf{i}+2\pi\epsilon(\mu_1 a'(0)-\mathbf{i}\delta)+\mathcal{O}(\epsilon^2)} + \mathcal{O}(\epsilon^2)$$

$$(3) : \xi_2(T) = e^{-2\pi\mathbf{i}+2\pi\epsilon(\mu_1 a'(0)+\mathbf{i}\delta)+\mathcal{O}(\epsilon^2)} + \mathcal{O}(\epsilon^2)$$

Per tant, arribem així a les equacions de bifurcació:

$$\Gamma_1 := \frac{1}{2\pi\varepsilon}(\xi_1(T) - 1) = \mathbf{i}\delta + a'(0)\mu_1 + \mathcal{O}(\varepsilon) = 0$$

$$\Gamma_2 := \frac{1}{2\pi\varepsilon}(\xi_2(T) - 1) = -\mathbf{i}\delta + a'(0)\mu_1 + \mathcal{O}(\varepsilon) = 0$$

$$\Gamma := \eta(T) - \eta_0 = (e^{2\pi B_0} - I)\eta_0 + \mathcal{O}(\varepsilon) = 0$$

.

Finalment, concloem que per $\epsilon = 0$, aquest sistema té solució única $\delta = 0$, $\mu_1 = 0$ i $\eta_0 = 0$.

Veiem que el següent Jacobià:

$$\partial(\Gamma_1, \Gamma_2, \Gamma)/\partial(\delta, \mu, \eta_0) = \begin{pmatrix} i & a'(0) & 0 \\ -i & a'(0) & 0 \\ 0 & 0 & e^{2\pi B_0} - Id \end{pmatrix}$$

té rang n per $\epsilon = 0$, ja que $a'(0) \neq 0$. Així, si $|\epsilon| \neq 0$ el teorema de la funció implícita[29] implica que el sistema de la bifurcació d'equacions té solució $\delta(\epsilon), \mu_1(\epsilon), \eta_0(\epsilon)$. □

Ara ja podem demostrar el Teorema de Lyapunov, que és una conseqüència del Teorema de la Bifurcació de Hopf:

[29] **Teorema de la funció implícita.** Sigui A un obert de $\mathbb{R}^n \times \mathbb{R}^n$, $f : A \to \mathbb{R}^m$ una $f \in \mathcal{C}^k$ en A i $p = (a, b) \in A$ tal que $f(a, b) = 0$. Si $\frac{\partial(f_1,\ldots,f_m)}{\partial(y_1,\ldots,y_m)}(a, b) = det\Big(\frac{\partial f_j}{\partial y_i}(a, b)\Big)_{1\leq i,j\leq m} \neq 0$. Aleshores,

$\begin{cases} (1) \text{ existeixen dos entorns, } U \subseteq \mathbb{R}^n \text{ de } a \text{ i } V \subseteq \mathbb{R}^m \text{ de } b, \text{ tal que } U \times V \subseteq A. \\ (2) \text{ existeix una funció } g : U \to V \text{ de classe } \mathcal{C}^k(U) \text{ tal que per a cada } x \in U \\ \quad y = g(x) \text{ és l'únic punt de } V \text{ que compleix } f(x, y) = 0. \end{cases}$

Demostració. Considerem un el sistema modificat:

$$\dot{\mathbf{x}} = A\mathbf{x} + f(\mathbf{x}) + \mu \text{grad } I(\mathbf{x}) \tag{A.18}$$

I provarem que es verifiquen totes les hipòtesis del Teorema de Hopf i que les òrbites periòdiques no estacionàries només passen si $\mu = 0$.

Per veure que les òrbites periòdiques no estacionàries només passen si $\mu = 0$, evaluem dId/dt sobre les solucions de (A.18):

$$\frac{dId}{dt} =< \text{grad}I(\mathbf{x}), A\mathbf{x} + f(\mathbf{x}) + \mu \text{grad}I(\mathbf{x}) > =$$

$$< \text{grad}I(\mathbf{x}), A\mathbf{x} + f(\mathbf{x}) > + \mu < \text{grad}I(\mathbf{x}), \text{grad}I(\mathbf{x}) >= \mu \ \mid \text{grad}I(\mathbf{x}) \mid^2 .$$

La darrera igualtat es compleix perquè $I(x)$ és una integral per (4.3). A més a més, $1/\mu dId/dt$ és una funció monòtona creixent tret que $\text{grad}I(x) = 0$, que dona $x(t) = 0$ que és un punt estacionari, perquè l'equació $\text{grad}I(x) = 0$ té $x = 0$ com a solució aïllada.

Per poder aplicar el teorema de Hopf, vegem com es comporta la part real del valor propi proper a $\mathbf{i}\beta$. Fent una transformació lineal, obtenim:

$$A = \begin{pmatrix} 0 & \beta & 0 \\ -\beta & 0 & 0 \\ 0 & 0 & \hat{A} \end{pmatrix},$$

on $\hat{A} \in \mathbb{R}^{(n-2)\times(n-2)}$ és una matriu quadrada real.

Al ser $I(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T S\mathbf{x} + o(|\mathbf{x}|^2)$ integral primera, $0 =< \text{grad}I(\mathbf{x}), A\mathbf{x} + f(\mathbf{x}) >$, així $\mathbf{x}^T SA\mathbf{x} = 0$. $A^T S + SA = 0$, així sabem la forma de S :

$$S = \begin{pmatrix} a & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & \hat{S} \end{pmatrix}$$

amb $a \neq 0$ ja que $detS \neq 0$. De la matriu

$$A + \mu S = \begin{pmatrix} \mu a & \beta & 0 \\ -\beta & \mu a & 0 \\ 0 & 0 & \hat{A} + \mu\hat{S} \end{pmatrix}$$

tenim que el valor proper a $\mathbf{i}\beta$ té part real $\alpha(\mu) = a\mu$ i, per tant, $\alpha'(0) = a \neq 0$ i així, ja es pot aplicar el Teorema de Hopf i queda demostrat el Teorema de Lyapunov.

□

B Codi

Aquest és el codi que s'ha desenvolupat per a fer el treball. Calcula òrbites planes o verticals en els punts fixos del sistema Hamiltonià del RTBP.

Les funcions $four1$ i *svdcmp* són del *Numerical Recipes in C*[9].

Notem que en el codi no hi ha cap accent.

Main

```
#include "funcions.h"
int main() {
   //Declaracio de variables

   int         N;          //Num punts de la grid que s'usaran.
   int         NxDIM;
   int         N_2;
   int         punt_fix;   //quin punt fix agafem
   int         newton_conv;//quant triga a convergir el newton
   int         num_orb;

   real      * theta;  //Vector de la grid punts
   real      * Li;     //Li
   real      * K0;     //k(0)

   real      ** D2U;
   real      ** DX;          //Diferencial del camp.

   real      ** VAPs;        //Matriu de VAPs.
   real      ** VEPs;        //Matriu de VEPs
   real      ** KG;          //Funcio de la grid de punts.
   real      ** D_m;         //operador derivada

   real        wp;
   real        h0;
   real        h;
   real        hi;
   real        ham_li; //energia en Li

   FILE* arxiu = NULL;

   N = 128;
   N = nextPowerOfTwo(N);//Si N no es potencia de 2, fem que ho sigui
   NxDIM = DIM * N;
   N_2 = 2 * N;

   theta = Theta(N);      //Vector de la grid de punts tq theta[i]=i*P/N

   Li = crearVect(DIM);  //memoria per Li

   punt_fix = 1;
   if(punt_fix==1||punt_fix==2||punt_fix==3)//Punts col.lineals
   {
       punts_col(Li, newton(0, punt_fix)); //Li passem u.
       //Recordem que u=x-1+MU
```

```
46      }
47      else if (punt_fix == 4)     //Punts triangulars
48      {
49          punts_triang(Li, punt_fix);
50      }
51
52      printf("L[%d] es:\n", punt_fix);
53      imprimirVector(Li, 6);
54
55      D2U = seg_deriv_pot(Li);
56      DX = crearMat(DIM, DIM);
57      diferencialcamp(DX, D2U);
58
59      VAPs = crearMat(DIM, DIM);
60      matriuVAPs(D2U, VAPs);
61      alliberarMat(D2U, 3);
62
63      //Trobem els VEPs
64      VEPs = crearMat(DIM, DIM);
65      matriuVEPs(DX, VAPs, VEPs);
66      alliberarMat(DX, DIM);
67
68      //Calculem KG.
69      //Passem 2 i 3 per calcular orbites planes,
70      //perque son les columnes on hi ha els VEPs plans en la matriu VEPs.
71      //Passem 4 5 per calcular les orbites veritcals.
72      KG = crearMat(DIM, N_2);
73      Ktheta(KG, theta, N, VEPs, Li, 2, 3);
74
75      K0 = crearVect(DIM);
76
77      //En aquest cas fem servir K0=H0
78      for (int i = 0; i < DIM; i++) {
79          K0[i] = KG[i][0];
80      }
81
82      ham_li = hamiltonia(Li);
83      h = hamiltonia(K0);
84
85      //Mirem cap on creix l'energia
86      printf("hamilt L1 es %.16Le\n", ham_li);
87      printf("hamilt h0 es %.16Le\n", h);
88
89      free(Li);
90
91      wp = VAPs[2][3] / pi2;//Frequencia per a orbites planes.
92      //Per a orbites verticals, VAPs[5][4]
93      alliberarMat(VAPs, DIM);
94      alliberarMat(VEPs, DIM);
95
96      D_m = operador_derivada(N, N_2);
97
98      bool refinament = true;
99
100     real* derivG_h_neg = crearVect(NxDIM + 2);
101
102     derivG_h_neg[NxDIM] = 1.0L;      //(0,0,...,0,1,0)
103
104     real** DG;
```

```
105     //Es crea DG nomes una vegada aqui fora, aixi es reutilitzara memoria.
106     DG = crearMat(NxDIM + 2, NxDIM + 1);
107     real** KG0 = crearMat(DIM, N_2);
108     real wp0 = wp;
109
110     ////////////////////
111     ///////NEWTON///////
112     ////////////////////
113
114     real* deriv_x;//Pel refinament
115     newton_conv = newton_bestia(KG, &wp, h, D_m, N, DG);
116     if (newton_conv == 0) {
117         printf("NO ha convergit a la primera\n");
118         exit(1);
119     }
120
121     printf("\n\n\t\t\t**********************************************\n\n\n");
122     printf("\t\t\t HA CONVERGIT\n");
123     printf("\n\n\t\t\t**********************************************\n\n\n");
124
125     {//escrivim en un fitxer les infos!
126         num_orb = 0;
127         char nom_arxiu[20];
128         sprintf_s(nom_arxiu, sizeof(nom_arxiu), "orbita%d.txt", num_orb);
129
130         // obrim el fitx
131         FILE* arxiu = fopen(nom_arxiu, "w");
132         if (arxiu == NULL) {
133             printf("Error a l'obrir l'arxiu");
134             exit(num_orb);
135         }
136         fprintf(arxiu, "temps x y z\n");
137
138         for (int j = 0; j < N; j++) {
139             fprintf(arxiu, "%Le ", theta[j]);
140             fprintf(arxiu, "% .8Le % .8Le % .8Le % .8Le % .8Le % .8Le", KG[cx
        ][j], KG[cy][j], KG[cz][j], KG[cpx][j], KG[cpy][j], KG[cpz][j]);
141             fprintf(arxiu, "\n");
142         }
143
144         // tanquem
145         fclose(arxiu);
146     }
147
148     //KG0=KG
149     for (int i = 0; i < DIM; i++)
150     {
151         memcpy(KG0[i], KG[i], N_2 * sizeof(real));
152     }
153
154     wp0 = wp;
155     h0 = h;
156     hi = 1E-3L;
157
158     if (refinament)//Es refina
159     {
160         //Pel refinament es fa servir el teorema de a funcio implicita,
161         //per trobar x'(h0) i obtenir una x m\'es bona.
162
```

```
163         //S'ha de calcular x'(h0) de
164         // DG(x_0, h0)x'(h0)=-\fracc{\Delta G}{\delta h}(x_0, h0)
165         //Necessitem un vector (0,...,0,1,0) de dimensions  NxDIM + 2
166 
167         //Ara resolem DG*x'(h0)=derivG_h_neg:
168         deriv_x = resol_sist_lin_amb_SVD(DG, derivG_h_neg, NxDIM + 2, NxDIM +
      1);
169 
170          //x'(h0) com a vector refinament, ho apliquem a la correccio
171 
172         //S'inicialitzen h, x=(KG,w).
173         int jxDIM = 0;
174         for (int j = 0; j < N; j++)
175         {
176             jxDIM = j * DIM;
177             for (int i = 0; i < DIM; i++)
178                 KG[i][j] += deriv_x[jxDIM + i] * hi;
179         }
180 
181         wp = wp + deriv_x[NxDIM + 1] * hi;
182         free(deriv_x);
183     }
184 
185     //Bucle per trobar una fam de num_orb \'orbites peri\'odiques.
186     do {
187         //S'inicialitzen h=h0+hi, x=x0.
188 
189         h = h0 + hi;//avancem un pas en la h
190 
191         for (int i = 0; i < DIM; i++)
192         {
193             memcpy(KG[i], KG0[i], N_2 * sizeof(real));
194         }
195         wp = wp0;
196 
197         newton_conv = newton_bestia(KG, &wp, h, D_m, N, DG);
198 
199         if (newton_conv == 0)//No ha convergit!
200         {
201             printf("\n\n\t\t\t************************************\n\n\n");
202             printf("\t\t\t NO HA CONVERGIT\n");
203             printf("\n\n\t\t\t************************************\n\n\n");
204 
205             hi = hi / 2;
206         }
207         else //Convergeix:)
208         {
209             num_orb++;
210             printf("\n\n\t\t\t************************************\n\n\n");
211             printf("\t\t\t HA CONVERGIT\n");
212             printf("\n\n\t\t\t************************************\n\n\n");
213 
214             {//escrivim en un fitxer les infos!
215 
216                 char nom_arxiu[20];
217                 sprintf_s(nom_arxiu, sizeof(nom_arxiu), "orbita%d.txt",
218                  num_orb);
219 
220                 //S'obre el fitxer
```

```
221                 FILE* arxiu = fopen(nom_arxiu, "w");
222                 if (arxiu == NULL) {
223                     printf("Error a l'obrir l'arxiu");
224                     exit(num_orb);
225                 }
226 
227                 fprintf(arxiu, "temps x y z px py pz \n");
228 
229 
230                 for (int j = 0; j < N; j++) {
231                     fprintf(arxiu, "%Le ", theta[j]);
232                     fprintf(arxiu, "% .8Le % .8Le % .8Le % .8Le % .8Le
233                             % .8Le", KG[cx][j], KG[cy][j], KG[cz][j],
234                             KG[cpx][j], KG[cpy][j], KG[cpz][j]);
235                     fprintf(arxiu, "\n");
236                 }
237 
238                 // tanquem l'arxiu
239                 fclose(arxiu);
240             }
241 
242             if(newton_conv <= 2)//Ha convergit molt r\'apid!
243                 hi = hi * 2;
244 
245             //KG0=KG
246             printf("haurien de ser una mica diferents!\n");
247             printf("KG0[3][25]=%.10Le\n", KG0[3][25]);
248             printf("KG=%.10Le\n", KG[3][25]);
249 
250             for (int i = 0; i < DIM; i++)
251             {
252                 memcpy(KG0[i], KG[i], N * sizeof(real));
253             }
254 
255             wp0 = wp;
256             h0 = h;
257         }
258 
259         ////////////////////////////////////////
260         //   Millorem la llavor (refinem):    //
261         ////////////////////////////////////////
262 
263         //Amb refinament. Si ha convergit,
264         if (refinament && newton_conv > 0)
265         {
266             //Ara resolem DG*x'(h0)=derivG_h_neg:
267             deriv_x = resol_sist_lin_amb_SVD(DG, derivG_h_neg, NxDIM + 2,
    NxDIM + 1);
268             //x'(h0) com a vector refinament, ho apliquem a la correccio!
269             //KG+x'(h0)[el q toqui]*hi.
270 
271             //Actualitzem KG
272             int jxDIM = 0;
273             for (int j = 0; j < N; j++)
274             {
275                 jxDIM = j * DIM;
276                 for (int i = 0; i < DIM; i++)
277                     KG[i][j] += deriv_x[jxDIM + i] * hi;
278             }
```

```
279             //Actualitzem  w.
280             wp = wp + deriv_x[NxDIM + 1] * hi;
281             free(deriv_x);
282         }
283
284     } while (num_orb<50);  //Creem 50 orbites
285
286     printf("Arriba fins a h = %.9Le\n", h);
287
288     //Alliberem memoria
289     free(theta);
290     alliberarMat(D_m, NxDIM);
291     alliberarMat(KG, DIM);
292     alliberarMat(KG0, DIM);
293     free(derivG_h_neg);
294     free(K0);
295     alliberarMat(DG, NxDIM + 2);
296
297     return 0;
298 }
```

Codi 1: Main del programa

Fitxer de capçalera

```
1  typedef long double real;
2
3  #include <stdio.h>
4  #include <string.h>
5  #include <stdbool.h>
6  #include <stdlib.h>
7  #include <math.h>
8
9  #define EPSILON    0.000001L   // Tolerancia newton dimensio 1
10 #define tol_zero   1.0E-18L      // considerem 0
11 #define MU       1.215058560962404E-2L   //MU del sistema Terra-Lluna
12 #define DIM      6
13 #define PERIOD     1.0L            //Periode de la funcio k.
14 #define R      1.0E-2L     //Radi al voltant del punt L_i
15 #define pi2      (8.0L*atanl(1))
16 #define tol_newton   1.0E-7L     //Tolerancia pel newton
17
18 #define cx 0
19 #define cpx 1
20 #define cy 2
21 #define cpy 3
22 #define cz 4
23 #define cpz 5
24
25 #define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr
26
27 #define SIGN(a,b) ((b) > 0.0 ? fabsl(a) : - fabsl(a))
28
29 static double maxarg1, maxarg2;
30 #define FMAX(a,b) (maxarg1 = (a),maxarg2 = (b),(maxarg1) > (maxarg2) ? (
       maxarg1) : (maxarg2))
31
```

```
32 static int iminarg1, iminarg2;
33 #define IMIN(a,b) (iminarg1 = (a),iminarg2 = (b),(iminarg1 < (iminarg2) ? (
       iminarg1) : iminarg2))
34
35 static double sqrarg;
36 #define SQR(a) ((sqrarg = (a)) == 0.0 ? 0.0 : sqrarg * sqrarg)
```

Codi 2: *Includes* i *defines*

```
1  //Producte de kronecker entre dues matrius
2  real** producte_kronecker(real** A, int fil_A, int col_A, real** B, int
       fil_B, int col_B);
3
4  //Derivada segona del potencial del Ham
5  real** seg_deriv_pot(real* qp);
6
7  //Calcula l'operador derivada discretitzat partint d'una base ortonormal
8  real** operador_derivada(int N, int N2);
9
10
11 //Sustitucio enrere, resol sistemes triangulars
12 real* backSubstitution(real** A, int n);
13
14 //Vector theta, vector de la grid de punts tq theta[i]=i*P/N
15 real* Theta(int N);
16
17 //Retorna la projeccio a l'eix k
18 real* proj_eix_k(int eix, int m);
19
20 //Resol un sistema lineal de la forma Ax=b fent la SVD
21 real* resol_sist_lin_amb_SVD(real** A, real* b, int fil, int col);
22
23 //Derivada Hamiltonia respecte el punt avaluat en el camp
24 real* deriv_hamiltonia_en_camp(real* X);
25 //Definir el potencial amb les derivades.
26 real Upot(real* qp, real* DU, real** DU2);
27
28 //Newton unidimensional de f(x)=0.
29 real newton_per_deriv_pot(real x0, int i);
30
31 //Hamiltonia en el punt qp
32 real hamiltonia(real* qp);
33
34 // Funcions de L1,L2,L3
35 real f(real u, int i);
36
37 // Derivada de les funcions en L1, L2 i L3
38 real df(real u, int i);
39
40 //Calcula sqrt( a^2 + b^2 ) amb una precisio decent.
41 //Extreta del Numerical Recipies
42 real pythag(real a, real b);
43
44 //Pel calcul de VAPs, B i C.
45 real B(real** DU2);
46 real C(real** DU2);
47
48 real R12(real*);     //r_1^2
49 real R22(real*);     //r_2^2
```

```
50
51 // Punts col.lineals: L1, L2, L3 passant - li la x corresponent
52 void punts_col(real* L, real x);
53
54 //Punts triangulars L4, L5. on punt=4 o 5
55 void punts_triang(real* Li, int punt);
56
57 //Diferencial del camp
58 void diferencialcamp(real**, real**);
59
60 //Matriu VAPs sense imaginaris
61 void matriuVAPs(real** DU2, real** VAPS);
62
63 //Martiu VEPs (v_1, v_2, u_p, v_p, u_v, v_v).
64 void matriuVEPs(real** DX, real** VAPs, real** VEPs);
65
66 // Eliminacio gaussiana, triangula la matriu
67 void gaussianElimination(real** A, int n);
68
69 //reordena com conve la matriu
70 void matriu_reordenada_veps_complexos(real** A, int m);
71
72 //Aqui es calcula el camp respecte un vector del camp: X(qp).
73 void campRTBP(real* qp, real* X);
74
75 //Fem servir el metode de Gauss per a trobar els Veps.
76 void Gauss_Veps(real **A, real VAP, real *vep, int n);
77
78 //Veps complexos amb les condicions del problema
79 void Gauss_Veps_complex(real **A, real VAP, real *vep, int n, real ** VEPs);
80
81 //Veps complexos en general
82 void Gauss_Veps_complex_general(real **A, real VAP_real, real VAP_imag, real
       *vep_re, real *vep_im, int n, real ** VEPs);
83
84 //Omple la matriu a la grid de punts, avaluant tots els punts a k.
85 void Ktheta(real** KG, real* theta, int N, real** VEPs, real* L1, int col1,
      int col2);
86
87 /*Calcula el camp en la grid de punts.
88     Omple XKG i la part de DG que conte les diferencials
89     en els diferents punts de la grid.*/
90 void camp_sistema_en_grid(real** KG, real** XKG, real** DG, int N);
91
92 /*
93   Rutina del Numerical Recipies.
94    Substitueix data[0,...2*nn-1] per la seva DFT si isign=1
95    o per la seva DFT inversa si isign=-1
96    nn HA DE SER una potencia de 2
97    En el nostre cas, adaptem l'ordre, sent fidels al codi del
98    Numerical Recipies, per a que quedi com ens conve:
99    *data te el format data[0]=Re(point0), data[nn-1+0]=Im(point0)
100   data[1]=Re(point1), data[nn-1+1]=Im(point1)
101   data[2]=Re(point2), data[nn-1+2]=Im(point2)
102   ...
103   data[2*j]=Re(pointj), data[nn-1+j]=Im(pointj)
104 */
105 void four1(real* data, int nn, int isign);
106
```

```
107 //Derivada discretitzada de data, en fourier
108 void iderivativeF(real* data, int nn);
109
110 //Passa data de grid a Fourier
111 void iG2F(real* data, int nn);
112
113 //Passa data de fourier a grid
114 void iF2G(real* data, int nn);
115
116
117 /*Fa tot el Newton. Mira si convergeix amb les ondicions inicials donades
        (KG, wp, h0) i, si ho fa, retorna sobre KG wp el resultat obtingut.*/
118 int newton_bestia(real** KG, real * wp, real h0, real **D_m, int N, real **
    DG);
119
120 //Retorna la seguent potencia de dos.
121 int nextPowerOfTwo(int n);
122
123 /*Descomposicio svd d'una matriu.
124   Modificada del Numerical Recipes in C:
125   Donada una matriu a[nRows][nCols], svdcmp() calcula la seva SVD compacta
126   A = U * Sigma * Vt. La matriu A es substituida per U,
127   la matriu diagonal Sigma surt com un vector w[ncols]
128   V (not la V transposada) es una matriu V[nCols][nCols].*/
129 int svdcmp(real** a, int nRows, int nCols, real* w, real** v);
130
131 //Adaptem l'ordre de four1 per ser fidels al codi extret de Numerical
    Recipies
132 int num_com_toca_ordre_meu(int num, int nn);
133 int num_com_toca_deriv(int num, int nn);
134
135 //Test
136 void provatura_hamilt_dif(void);
137
138 //Funcions generiques, nom descriptiu
139 real  ** mult_mat_mat     (real** matriu1, real** matriu2, int files1, int
    col1, int files2, int col2);
140 real  ** mult_mat_escal   (real** mat, real escal, int files, int col);
141 real  ** restar_mat_mat   (real** matriu1, real** matriu2, int files1, int
    col1, int files2, int col2);
142 real  ** crearMat         (int files, int columnes);
143 real  ** crear_mat_identitat  (int files, int columnes);
144
145 real  * mult_mat_vect     (real** matriu, real* vect, int files, int
    columnes);
146 real  * mult_escal_vect   (real escal, real* vect, int files);
147 real  * restar_vect     (real* v1, real* v2, int files);
148 real  * crearVect       (int n);
149 real  * restarVect      (real* v1, real* v2, int n);
150
151 real  mult_dos_vect_a_real  (real* v1, real* v2, int n);
152 real  max_vec_abs       (real* arr, int n);
153 real  max_mat_abs       (real** vec, int fil, int col);
154
155 void  inv_vect_diag     (real* mat, int files);
156 void  transpose_square_matrix (real** A, real** B, int N);
157 void  imprimirVector      (real* vec, int n);
158 void  imprimir_vector_en_arxiu(real* vec, int n, FILE* arxiu_error);
159 void  imprimirMatriu      (real** A, int n, int m);
```

```
160 void  intercanviar_columnes (real** matriu, int columna_n, int columna_m,
    int N);
161 void  assignar_vect_a_col   (real** matriu, real* vector, int columna, int
    files);
162 void  assignar_vect_a_fila  (real** matriu, real* vector, int fila, int
    columnes);
163 void  reordenar_cols      (real** A, int* ordre_nou, int fil, int col);
164 void  alliberarMat      (real** matriu, int files);// Funcio que allibera la
     memoria assignada per a la matriu
165
166 //s'ha fet servir en test
167 void  imprimir_vector_en_dues_col_arxiu(real *vec,int n, FILE *arxiu_error);
168
169 void    imprimirMatriu_col_reals_col_im(real** A, int n, int m, FILE* arxiu)
    ;
```

Codi 3: Declaració i explicació de funcions

Fitxer *funcions.c*

```
1     #include "funcions.h"
2
3 real** producte_kronecker(real** A, int fil_A, int col_A, real** B, int
    fil_B, int col_B)
4 {
5     real** C = crearMat(fil_A * fil_B, col_A * col_B);
6     {
7         int i_filB, j_colB;
8
9         for (int i = 0; i < fil_A; i++) {
10
11            i_filB = i * fil_B;
12
13            for (int j = 0; j < col_A; j++) {
14
15                j_colB = j * col_B;
16
17                for (int k = 0; k < fil_B; k++) {
18
19                    for (int l = 0; l < col_B; l++) {
20
21                        // Cada element de la matriu A es multiplica per
    TOTA la matriu B
22                        C[i_filB + k][j_colB + l] = A[i][j] * B[k][l];
23
24                    }
25                }
26            }
27        }
28    }
29    return C;
30 }
31
32 real** seg_deriv_pot(real* qp)
33 {
34    real** D2U;
35    {
```

```
36          D2U = crearMat(3, 3);
37          real x = qp[0];
38          real y = qp[2];
39          real z = qp[4];
40
41          real r12 = R12(qp);//r_1^2
42          real r22 = R22(qp);//r_2^2
43
44          real r13 = r12 * sqrtl(r12);        //r_1^3=r_1^2*r_1
45          real r23 = r22 * sqrtl(r22);        //r_2^3=r_2^2*r_2
46
47          real r15 = r13 * r12;       //r_1^5=r_1^3*r_1^2
48          real r25 = r23 * r22;       //r_2^5=r_2^3*r_2^2
49
50          //
51          // Segones derivades de U:
52          //
53
54          //D_xx
55          D2U[0][0] = -1.0L + (1.0L - MU) / r13 + MU / r23 - 3.0L * ((1.0L -
    MU) * (x + MU) * (x + MU)) / r15 - 3.0L * MU * ((x - (1.0L - MU)) * (x -
    (1.0L - MU))) / r25;
56
57          //D_xy, D_yx
58          D2U[0][1] = D2U[1][0] = -3.0L * y * (((1.0L - MU) * (x + MU)) / r15
    + (MU * (x - (1.0L - MU))) / r25);
59
60          //D_xz, D_zx
61          D2U[0][2] = D2U[2][0] = -3.0L * z * (((1.0L - MU) * (x + MU)) / r15
    + (MU * (x - (1.0L - MU))) / r25);
62
63          //D_yy
64          D2U[1][1] = -1.0L + (1.0L - MU) / r13 + MU / r23 - 3.0L * y * y *
    ((1.0L - MU) / r15 + MU / r25);
65
66          //D_yz, D_zy
67          D2U[1][2] = D2U[2][1] = -3.0L * y * z * ((1.0L - MU) / r15 + MU /
    r25);
68
69          //D_zz
70          D2U[2][2] = (1.0L - MU) / r13 + MU / r23 - 3.0L * z * z * ((1.0L -
    MU) / r15 + MU / r25);
71      }
72      return D2U;
73  }
74
75  real** operador_derivada(int N, int Nx2)
76  {
77      real** D_m;
78      {
79          real** I_N;
80          real** I_N_four;
81          real** DI_N_four;
82          real** DI_N;
83          real** I_m;
84
85          real aux;
86
87          I_N = crearMat(N, Nx2);//Matriu identitat.
```

```
88          for (int i = 0; i < N; i++)
89              I_N[i][i] = 1.0L;
90
91          //Calculem Dbarret amb Fourier:
92          I_N_four = crearMat(N, Nx2);
93          DI_N_four = crearMat(N, Nx2);
94          DI_N = crearMat(N, Nx2);
95          for (int i = 0; i < N; i++)
96          {
97              memcpy(I_N_four[i], I_N[i], Nx2 * sizeof(real));
98              iG2F(I_N_four[i], N);            // I_N_four = FFT(I_N).
99              memcpy(DI_N_four[i], I_N_four[i], Nx2 * sizeof(real));
100             iderivativeF(DI_N_four[i], N);   // DI_N_four = derivada(
    I_N_four).
101             memcpy(DI_N[i], DI_N_four[i], Nx2 * sizeof(real));
102             iF2G(DI_N[i], N);            // D_barret = IFFT(DI_N_four).
103         }
104
105         //Alliberem memoria de les matrius que ja no necessitem.
106         alliberarMat(I_N, N);
107         alliberarMat(I_N_four, N);
108         alliberarMat(DI_N_four, N);
109
110         //Transposem la matriu, que sera l'operador
111         //Ja que la derivada s'ha calculat sobre files i no sobre columnes
112         for (int i = 0; i < N; i++) {
113             for (int j = 0; j < i; j++)
114             {
115                 aux = DI_N[i][j];
116                 DI_N[i][j] = DI_N[j][i];
117                 DI_N[j][i] = aux;
118             }
119         }
120
121         I_m = crear_mat_identitat(DIM, DIM);
122         //Trobem l'operador amb el producte de Kronecker
123         D_m = producte_kronecker(DI_N, N, N, I_m, DIM, DIM);
124         alliberarMat(I_m, DIM);
125         alliberarMat(DI_N, N);
126     }
127     return D_m;
128 }
129
130 real* backSubstitution(real** matriu, int m)
131 {
132     real* x = crearVect(m);
133
134     {
135         int n = m + 1;
136         real** A = crearMat(m, n);
137
138         for (int i = 0; i < m; i++) {
139             for (int j = 0; j < m; j++) {
140                 A[i][j] = matriu[i][j];
141             }
142         }
143
144         int index = 0;
145         if (fabsl(A[m - 1][m - 1]) <= 0.1E-10 && (fabsl(A[m - 2][m - 2]) >=
```

```
    0.1E-10)) {
146             x[m - 1] = 1.0L;
147             for (int i = 0; i < m; i++)//Tota la ultima fila de zeros!
148                 A[m - 1][i] = 0.0L;
149             index = 2;
150         }
151         else if ((fabsl(A[m - 1][m - 1]) <= 0.1E-10) && (fabsl(A[m - 2][m -
    2]) <= 0.1E-10))
152         {
153             x[m - 1] = 0.0L;
154             x[m - 2] = 1.0L;
155             //A[m - 1][m - 1] = 0.0L;
156             index = 3;
157         }
158         else {
159             x[m - 1] = A[m - 1][m] / A[m - 1][m - 1];
160             index = 2;
161         }
162 
163         for (int i = m - index; i >= 0; i--) {
164             real sum = 0.0L;
165             for (int j = i + 1; j < m; j++) {
166                 sum += A[i][j] * x[j];
167             }
168             x[i] = (A[i][m] - sum) / A[i][i];
169         }
170         alliberarMat(A, m);
171     }
172 
173     return x;
174 }
175 
176 real* Theta(int N)
177 {
178     real* theta;
179     {
180         theta = crearVect(N);
181         for (int i = 0; i < N; i++)
182         {
183             theta[i] = i * PERIOD / N;
184         }
185     }
186     return theta;
187 }
188 
189 real* proj_eix_k(int eix_k, int m)
190 {
191     real* proj;
192     {
193         proj = crearVect(m);
194         proj[eix_k] = 1.0L;
195     }
196     return proj;
197 }
198 
199 real* resol_sist_lin_amb_SVD(real** A, real* b, int fil_DG, int col_DG)
200 {
201     real* vector_solucio;
202     {
```

```
203         //Resoldre amb SVD el seguent: Ax=b.
204         real* sigma = crearVect(col_DG);
205 
206         real** V = crearMat(col_DG, col_DG);
207         real** U_T = crearMat(col_DG, fil_DG);
208         real** U = crearMat(fil_DG, col_DG);
209         //U=A
210         for (int i = 0; i < fil_DG; i++)
211             memcpy(U[i], A[i], col_DG * sizeof(real));
212 
213         svdcmp(U, fil_DG, col_DG, sigma, V);
214 
215         //Test. Comprovem que svdcmp esta ben fet: A=U*sigma*V^T?
216         {
217             real* err_svd = crearVect(fil_DG * col_DG);
218             real err_max_svd;
219             real** V_trans = crearMat(col_DG, col_DG);
220 
221             transpose_square_matrix(V, V_trans, col_DG);
222             //multipliquem sigma*V^T i la guardem a V_trans:
223             real aux_sum = 0.0L;
224 
225             for (int i = 0; i < col_DG; i++)
226                 for (int j = 0; j < col_DG; j++)
227                     V_trans[i][j] = sigma[i] * V_trans[i][j];
228             //Ara multipliquem per U.
229             real** A_aux2;
230             A_aux2 = mult_mat_mat(U, V_trans, fil_DG, col_DG, col_DG, col_DG
    );
231 
232             printf("resta per comprovar que lsvdcmp es fa be.\n");
233             for (int i = 0; i < fil_DG; i++)
234                 for (int j = 0; j < col_DG; j++)
235                 {
236                     err_svd[i * col_DG + j] = A_aux2[i][j] - A[i][j];
237                 }
238             err_max_svd = max_vec_abs(err_svd, fil_DG * col_DG);
239             printf("err_max_svd=%Le\n", err_max_svd);
240             //La comprovacio acaba aqui.
241             alliberarMat(A_aux2, fil_DG);
242             alliberarMat(V_trans, col_DG);
243             free(err_svd);
244         }
245 
246         inv_vect_diag(sigma, col_DG);
247         printf("ultima sigma=%Le\n", sigma[col_DG - 1]);
248         //V=V*sigma^-1:
249         for (int i = 0; i < col_DG; i++)
250             for (int j = 0; j < col_DG; j++)
251                 V[i][j] = V[i][j] * sigma[j];
252 
253         for (int i = 0; i < col_DG; i++)
254             for (int j = 0; j < fil_DG; j++)
255                 U_T[i][j] = U[j][i];
256 
257         real* U_T_per_b;
258 
259         //U_T_per_b=U^T*b
260         U_T_per_b = mult_mat_vect(U_T, b, col_DG, fil_DG);
```

```
261         //sol :V*sigma^(pseudoinversa)*U^T*b
262         vector_solucio = mult_mat_vect(V, U_T_per_b, col_DG, col_DG);
263 
264         //Calculem l'error del sistema lineal. i.e. A*x-b==0???
265         {
266             //Farem DG*vector_sol-b;
267             real* error_sist_lineal;
268             real err_max_sis_lin;
269             error_sist_lineal = mult_mat_vect(A, vector_solucio, fil_DG,
    col_DG);
270             for (int i = 0; i < fil_DG; i++) {
271                 error_sist_lineal[i] -= b[i];
272             }
273             err_max_sis_lin = max_vec_abs(error_sist_lineal, fil_DG);
274             printf("L'error maxim del sist lineal DG*vector_sol-b es: %Le\n"
    , err_max_sis_lin);
275             free(error_sist_lineal);
276             printf("Mida de b, dins la funcio es: % Le\n", max_vec_abs(b,
    fil_DG));
277             printf("Mida de sol  (increments), dins la funcio es: % Le\n",
    max_vec_abs(vector_solucio, col_DG));
278             //imprimirVector(vector_solucio, col_DG);
279         }
280 
281         //Alliberem memoria
282         free(sigma);
283         alliberarMat(V, col_DG);
284         alliberarMat(U_T, col_DG);
285         alliberarMat(U, fil_DG);
286         free(U_T_per_b);
287     }
288     return vector_solucio;
289 }
290 
291 real* deriv_hamiltonia_en_camp(real* X)
292 {
293     real* sol = crearVect(DIM);
294     {
295         //venint del camp:
296         sol[0] = -X[1];
297         sol[1] = X[0];
298         sol[2] = -X[3];
299         sol[3] = X[2];
300         sol[4] = -X[5];
301         sol[5] = X[4];
302     }
303     return sol;
304 }
305 
306 real Upot(real* qp, real* DU, real** DU2)
307 {
308     real U;
309     {
310         real x = qp[0];
311         real y = qp[2];
312         real z = qp[4];
313 
314         real r12 = R12(qp);//r_1^2
315         real r22 = R22(qp);//r_2^2
```

```
316
317         real r13 = r12 * sqrtl(r12);       //r_1^3=r_1^2*r_1
318         real r23 = r22 * sqrtl(r22);       //r_2^3=r_2^2*r_2
319
320         real r15 = r13 * r12;      //r_1^5=r_1^3*r_1^2
321         real r25 = r23 * r22;      //r_2^5=r_2^3*r_2^2
322
323         U = -0.5L * (x * x + y * y) - (1.0L - MU) / sqrtl(r12) - MU / sqrtl(
    r22);
324
325         //
326         //Primeres derivades de U:
327         //
328
329         //D_x
330         DU[0] = -x + ((1.0L - MU) * (x + MU)) / r13 + (MU * (x - (1.0L - MU)
    )) / r23;
331
332         //D_y
333         DU[1] = (-1.0L + (1.0L - MU) / r13 + MU / r23) * y;
334
335         //D_z
336         DU[2] = ((1.0L - MU) / r13 + MU / r23) * z;
337         for (int i = 0; i < 3; i++)
338         {
339             if (fabsl(DU[i]) < tol_zero)
340                 DU[i] = 0.0L;
341         }
342
343         //
344         // Segones derivades de U:
345         //
346
347         //D_xx
348         DU2[0][0] = -1.0L + (1.0L - MU) / r13 + MU / r23 - 3.0L * ((1.0L -
    MU) * (x + MU) * (x + MU)) / r15 - 3.0L * MU * ((x - (1.0L - MU)) * (x -
    (1.0L - MU))) / r25;
349
350         //D_xy, D_yx
351         DU2[0][1] = DU2[1][0] = -3.0L * y * (((1.0L - MU) * (x + MU)) / r15
    + (MU * (x - (1.0L - MU)))) / r25);
352
353         //D_xz, D_zx
354         DU2[0][2] = DU2[2][0] = -3.0L * z * (((1.0L - MU) * (x + MU)) / r15
    + (MU * (x - (1.0L - MU)))) / r25);
355
356         //D_yy
357         DU2[1][1] = -1.0L + (1.0L - MU) / r13 + MU / r23 - 3.0L * y * y *
    ((1.0L - MU) / r15 + MU / r25);
358
359         //D_yz, D_zy
360         DU2[1][2] = DU2[2][1] = -3.0L * y * z * ((1.0L - MU) / r15 + MU /
    r25);
361
362         //D_zz
363         DU2[2][2] = (1.0L - MU) / r13 + MU / r23 - 3.0L * z * z * ((1.0L -
    MU) / r15 + MU / r25);
364     }
365     return U;
```

```
366 }
367
368 real newton_per_deriv_pot(real x0, int i) {
369     //Li passem u. Recordem que u=x-1+MU
370     //Aixi doncs, x=u+1-MU
371     real x1;
372     {
373         x1 = x0;
374         real error = 1.0L;
375         int iter = 0;
376         while (error > EPSILON && iter < 100000) {
377             real y = x1 - f(x1, i) / df(x1, i);
378             error = fabsl(y - x1);
379             x1 = y;
380             iter++;
381         }
382         //x=u+1-MU
383         x1 = x1 + 1 - MU;
384     }
385     return x1;
386 }
387
388 real hamiltonia(real* qp)
389 {
390     real sol;
391     {
392         real  x = qp[0];
393         real px = qp[1];
394         real  y = qp[2];
395         real py = qp[3];
396         real  z = qp[4];
397         real pz = qp[5];
398
399         real r12 = R12(qp);//r_1^2
400         real r22 = R22(qp);//r_2^2
401
402         real r13 = r12 * sqrtl(r12);       //r_1^3=r_1^2*r_1
403         real r23 = r22 * sqrtl(r22);       //r_2^3=r_2^2*r_2
404
405         real r15 = r13 * r12;      //r_1^5=r_1^3*r_1^2
406         real r25 = r23 * r22;      //r_2^5=r_2^3*r_2^2
407         real U_bar;
408
409         U_bar = -0.5L * (x * x + y * y) - (1.0L - MU) / sqrtl(r12) - MU /
    sqrtl(r22);
410
411         sol = 0.5 * ((px + y) * (px + y) + (py - x) * (py - x) + pz * pz) +
    U_bar;
412     }
413     return sol;
414 }
415
416 real f(real u, int i)
417 {
418     real resultat = 0.0L;
419     {
420         if (i == 1) {//L_1
421             resultat = MU * (u * u * u * u + 2.0L * u * u * u - u * u - 2.0L
     * u - 1.0L) - u * u * (u * u * u + 3.0L * u * u + 3.0L * u);
```

```
422         }
423         else if (i == 2) {//L_2
424             resultat = MU * (u * u * u * u + 2.0L * u * u * u + u * u + 2.0L
    * u + 1.0L) - u * u * (u * u * u + 3.0L * u * u + 3.0L * u);
425         }
426         else if (i == 3) {//L_3
427             resultat = MU * (u * u * u * u + 2.0L * u * u * u + u * u - 2.0L
    * u - 1.0L) - u * u * (u * u * u + 3.0L * u * u + 3.0L * u + 2.0L);
428         }
429         else {
430             printf("Enter mal posat a l'hora de calcular L[i]");
431             return 1;
432         }
433     }
434     return resultat;
435 }
436
437 real df(real u, int i)
438 {
439     real resultat = 0.0L;
440     {
441         if (i == 1) {//L_1
442             resultat = MU * (4.0L * u * u * u + 6 * u * u - 2.0L * u - 2.0L)
    - 2.0L * u * (u * u * u + 3.0L * u * u + 3.0L * u) - u * u * (3.0L * u *
    u + 6.0L * u + 3.0L);
443         }
444         else if (i == 2) {//L_2
445             resultat = MU * (4.0L * u * u * u + 6.0L * u * u + 2.0L * u +
    2.0L) - 2.0L * u * (u * u * u + 3.0L * u * u + 3.0L * u) - u * u * (3.0L
    * u * u + 6.0L * u + 3.0L);
446         }
447         else if (i == 3) {//L_3
448             resultat = MU * (4.0L * u * u * u + 6.0L * u * u + 2.0L * u -
    2.0L) - 2.0L * u * ((u * u * u + 3.0L * u * u + 3 * u + 2.0L)) - u * u *
    (3.0L * u * u + 6.0L * u + 3.0L);
449         }
450     }
451     return resultat;
452
453 }
454
455 real pythag(real a, real b) {
456     real absa, absb;
457
458     absa = fabsl(a);
459     absb = fabsl(b);
460
461     if (absa > absb)
462         return (absa * sqrtl(1.0L + SQR(absb / absa)));
463     else
464         return (absb == 0.0 ? 0.0 : absb * sqrtl(1.0L + SQR(absa / absb)));
465 }
466
467 real B(real** DU2)
468 {
469     real be;
470     {
471         be = 2.0L + (DU2[1][1] + DU2[0][0]) / 2.0L;
472     }
```

```
473     return be;
474 }
475 real C(real** DU2)
476 {
477     real ce;
478     {
479         ce = DU2[0][0] * DU2[1][1] - DU2[0][1] * DU2[0][1];
480     }
481     return ce;
482 }
483 
484 real R12(real* qp)
485 {
486     real r_12;
487     {
488         real x = qp[0];
489         real y = qp[2];
490         real z = qp[4];
491         r_12 = (x + MU) * (x + MU) + y * y + z * z;
492     }
493     return r_12;
494 }
495 
496 real R22(real* qp)//r_2^2
497 {
498     real r_22;
499     {
500         real x = qp[0];
501         real y = qp[2];
502         real z = qp[4];
503         r_22 = (x - (1.0L - MU)) * (x - (1.0L - MU)) + y * y + z * z;
504     }
505     return r_22;
506 }
507 
508 void punts_col(real* L, real x)
509 {
510     {
511         L[0] = x;
512         for (int i = 1; i < 6; i++)
513         {
514             if (i != 3)
515                 L[i] = 0.0L;        //Els punts col tenen y=z=0, pz=px=0 i py
    =x
516             //pque px+y=py-x=pz=0.
517         }
518 
519         L[3] = x;
520     }
521 }
522 
523 void punts_triang(real* Li, int punt)//pz=px i py=x: (x, -y, y, x, z, 0). Al
     trobar-se en el pla, z=0.
524 {
525     {
526         real x, y, z;
527 
528         x = 1.0L / 2.0L - MU;
529         y = sqrtl(3.0L) / 2.0L;
```

```
530         z = 0.0L;
531
532         if (punt == 5)//Per L5, y=-sqrtl(3.0L) / 2.0L.
533         {
534             y = -y;
535         }
536
537         Li[0] = x;
538         Li[1] = -y;
539         Li[2] = y;
540         Li[3] = x;
541         Li[4] = z;
542         Li[5] = 0.0L;
543     }
544 }
545
546 void diferencialcamp(real** DX, real** DU2)
547 {
548     /*x=(x,px,y,py,z,pz)*/
549     {
550         DX[0][1] = 1.0L;
551         DX[0][2] = 1.0L;
552
553         DX[1][0] = -1.0L - DU2[0][0];       //U_xx
554         DX[1][2] = -DU2[0][1];              //"U_xy"
555         DX[1][3] = 1.0L;
556         DX[1][4] = -DU2[0][1];              //"U_xz"
557
558         DX[2][0] = -1.0L;
559         DX[2][3] = 1.0L;
560
561         DX[3][0] = -DU2[1][0];               //-"U_yx"
562         DX[3][1] = -1.0L;
563         DX[3][2] = -1.0L - DU2[1][1];       //-1-"U_yy"
564         DX[3][4] = -DU2[1][2];              //-"U_yz"
565
566         DX[4][5] = 1.0L;
567
568         DX[5][0] = -DU2[2][0];              //-"U_zx"
569         DX[5][2] = -DU2[2][1];              //-"U_zy"
570         DX[5][4] = -DU2[2][2];              //-"U_zz"
571     }
572 }
573
574 void matriuVAPs(real** DU2, real** VAPS)
575 {
576     // Amb la mateix matriu de VAPs, necessitarem una matriu de VEPs q no
    inclogui nombres imaginaris.
577     // Aixi, en crearem una q sigui (v_1, v_2, u_p, v_p, u_v, v_v).
578     {
579         real Be = B(DU2);
580         real Ce = C(DU2);
581         real aux = sqrtl(Be * Be - Ce);
582         real lambda = sqrtl(-Be + aux);
583         real wp;
584         real w_v;
585
586         VAPS[0][0] = lambda;
587         VAPS[1][1] = -lambda;
```

```
588
589         wp = sqrtl(Be + aux);
590         VAPS[2][3] = -wp;
591         VAPS[3][2] = wp;
592
593         w_v = sqrtl(DU2[2][2]);
594         VAPS[4][5] = -w_v;
595         VAPS[5][4] = w_v;
596     }
597 }
598
599 void matriuVEPs(real** DX, real** VAPs, real** VEPs)
600 {//Fem una matriu on hi hagi els VEPs.
601     {
602         real** A_1 = crearMat(4, 4);
603         real* vector = crearVect(DIM);
604         int i, j;
605
606         for (i = 0; i < 4; i++)
607         {
608             for (j = 0; j < 4; j++)
609             {
610                 A_1[i][j] = DX[i][j];
611             }
612         }
613
614         for (i = 0; i < 2; i++)
615         {
616             Gauss_Veps(A_1, VAPs[i][i], vector, 4);//vector sera el VEP.
617             assignar_vect_a_col(VEPs, vector, i, DIM);
618         }
619         //Ara ja tenim els dos VEPs dels VAPs reals. Busquem els altres 4.
620         // Tinguem en compte q son complexos. trobem els veps per complexos
621
622         //VEPs plans:
623         Gauss_Veps_complex(A_1, VAPs[2][3], vector, 4, VEPs);//vector sera
    el VEP.
624
625         //Ara ja tenim els VEPs imaginaris plans.
626         // Anem a pels verticals, ja s'han trobat explicitament:
627         VEPs[4][4] = 1.0L;
628         VEPs[5][4] = 0.0L;
629         VEPs[4][5] = 0.0L;
630         VEPs[5][5] = VAPs[4][5];
631
632
633         printf("La matriu de VEPs es la seguent:\n");
634         imprimirMatriu(VEPs, 6, 6);
635
636         alliberarMat(A_1, 4);
637         free(vector);
638         //Test. comprovem que tenim els veps i els vaps be.
639         {
640             //Fem   DX*VEPs-VEPs*VAPs==0??.
641             for (int i = 0; i < 6; i++)
642             {
643                 for (int k = 0; k < 6; k++)
644                 {
645                     long double error;
```

```
646                     error = 0.0L;
647                     for (int j = 0; j < 6; j++)
648                     {
649                         error += DX[i][j] * VEPs[j][k] - VEPs[i][j] * VAPs[j
    ][k];
650                     }
651                     printf("ERROR veps i vaps: %Le\n", error);
652                 }
653             }
654         }
655     }
656 }
657
658 void gaussianElimination(real** A, int n)
659 {
660     {
661         int i, j, k;
662         real c;
663
664         for (j = 0; j < n; j++) {
665             for (i = 0; i < n; i++) {
666                 if (i > j && (fabsl(A[j][j]) >= 1.0E-16L)) {
667                     c = A[i][j] / A[j][j];
668                     for (k = 0; k < n; k++) {
669                         A[i][k] = A[i][k] - c * A[j][k];
670                     }
671                 }
672             }
673         }
674     }
675 }
676
677 void matriu_reordenada_veps_complexos(real** A, int m)
678 {
679     {
680         real** AUX;
681         AUX = crearMat(2 * m, 2 * m);
682         int i, j;
683         int parell = 0, senar = m;
684
685         //AUX=A
686         for (i = 0; i < 2 * m; i++)
687             for (j = 0; j < 2 * m; j++)
688                 AUX[i][j] = A[i][j];
689
690
691         for (j = 0; j < 2 * m; j++)
692         {
693             if (j % 2 == 0)
694             {
695                 for (i = 0; i < 2 * m; i++)
696                 {
697                     A[i][j] = AUX[i][parell];
698                 }
699                 parell++;
700             }
701             else
702             {
703                 for (i = 0; i < 2 * m; i++)
```

```
704                 {
705                     A[i][j] = AUX[i][senar];
706                 }
707                 senar++;
708             }
709         }
710         free(AUX);
711     }
712 }
713
714 void campRTBP(real* qp, real* X)
715 {
716     real* DU = crearVect(3);
717     real** DU2 = crearMat(3, 3);;
718     Upot(qp, DU, DU2);
719     {
720         real  x = qp[0];
721         real  y = qp[2];
722         real  z = qp[4]; // No cal redefinir-la perque no la fem servir.
723         real px = qp[1];
724         real py = qp[3];
725         real pz = qp[5];
726
727         X[0] = px + y;
728         X[1] = py - x - DU[0];   //py-x-U_x
729         X[2] = py - x;
730         X[3] = -px - y - DU[1]; //-px-y-U_y
731         X[4] = pz;
732         X[5] = -DU[2];           //-U_z
733     }
734 }
735
736 void Gauss_Veps(real** matriuATriang, real VAP, real* vep, int n)
737 {
738     {
739         if (matriuATriang == NULL || vep == NULL || n <= 0) {
740             printf("Entrada en Gauss_Veps no valida.\n");
741             return;
742         }
743         //real sum=0.0L;
744         real c = 0.0L;
745         real* aux;
746         real** A = crearMat(n, n);//Faig una matriu auxiliar perque NO VULL
    editar la matriu q em passen (matriu a Triangularitzar.)
747         for (int i = 0; i < n; i++)
748             for (int j = 0; j < n; j++)
749                 A[i][j] = matriuATriang[i][j];
750
751         for (int j = 0; j < n; j++) {
752             A[j][j] = A[j][j] - VAP;
753         }
754
755         // Matriu triangular superior
756         gaussianElimination(A, n);
757
758         // Ara nomes falta resoldre el sistema
759         // Posem v[n-1]=1. Perque es SCI. I a partir d'aqui anem fent.
760
761         aux = backSubstitution(A, n);
```

```
762
763         for (int i = 0; i < n; i++) {
764             A[i][i] = A[i][i] - VAP;
765         }
766
767         //Test
768         //real* per_comprovar = restar_vect(mult_mat_vect(A, aux, n, n),
    mult_escal_vect(VAP, aux, n), n);
769         //imprimirVector(per_comprovar, n);
770         if (n == 2)
771         {
772             for (int i = 0; i < 4; i++)
773             {
774                 vep[i] = 0.0L;
775             }
776             vep[4] = aux[0];
777             vep[5] = aux[1];
778
779         }
780         else if (n == 4)
781         {
782             for (int i = 0; i < 4; i++)
783             {
784                 vep[i] = aux[i];
785             }
786         }
787         else
788         {
789             printf("Oju perq el nostre Gauss_Veps esta programat nomes per
    per n=2 o n=4.\n");
790         }
791         free(aux);
792         alliberarMat(A, n);
793     }
794 }
795
796 void Gauss_Veps_complex(real** matriuATriang, real VAP, real* vep, int n,
    real** VEPs)
797 {
798     {
799         real* vep_aux;
800         vep_aux = crearVect(n);
801         Gauss_Veps_complex_general(matriuATriang, 0.0L, VAP, vep_aux, vep, n
    , VEPs);
802         free(vep_aux);
803     }
804 }
805
806 void Gauss_Veps_complex_general(real** matriuATriang, real VAP_real, real
    VAP_imag, real* vep_re, real* vep_im, int m, real** VEPs)
807 {
808     {
809         if (matriuATriang == NULL || m <= 0) {
810             printf("Entrada en Gauss_Veps no valida.\n");
811             return;
812         }
813
814         int i, j, k;
815         bool triangulized = false;
```

```
816         real c = 0.0L;
817         real* aux;
818         //Faig una matriu auxiliar perque NO VULL editar la matriu q em
    passen (matriu a Triangularitzar.)
819         real** A = crearMat(m * 2, m * 2);
820 
821         for (i = 0; i < m; i++)
822         {
823             for (j = 0; j < m; j++)
824             {
825                 if (i == j)
826                 {
827                     A[i][j] = matriuATriang[i][j] - VAP_real;
828                 }
829                 else
830                 {
831                     A[i][j] = matriuATriang[i][j];
832                 }
833             }
834             for (j = 0; j < m; j++)
835             {
836                 if (i == j) {
837                     A[i][m + j] = +VAP_imag;
838                 }
839             }
840         }
841 
842         for (i = 0; i < m; i++)
843         {
844             for (j = 0; j < m; j++)
845             {
846                 if (i == j) {
847                     A[m + i][j] = -VAP_imag;
848                 }
849             }
850             for (j = 0; j < m; j++)
851             {
852                 if (i == j)
853                 {
854                     A[m + i][m + j] = matriuATriang[i][j] - VAP_real;
855                 }
856                 else
857                 {
858                     A[m + i][m + j] = matriuATriang[i][j];
859                 }
860             }
861         }
862 
863         //Primer, es reordena la matriu per a que quedi real, im, re, im, re
    , im
864         matriu_reordenada_veps_complexos(A, m);
865 
866         // Matriu triangular superior
867         printf("La matriu triangular superior es la seguent:\n");
868         for (i = 0; i < 2 * m - 1; i++)
869         {
870             if (A[i][i] == 0) {
871                 // Buscar una fila sota per intercanviar
872                 for (k = i + 1; k < 2 * m; k++) {
```

```
873                     if (A[k][i] != 0) {
874                         // intercanviar fila i amb fila k
875                         for (j = i; j < 2 * m; j++) {
876                             real temp = A[i][j];
877                             A[i][j] = A[k][j];
878                             A[k][j] = temp;
879                         }
880                         break;
881                     }
882                 }
883                 if (A[i][i] == 0 && i >= 2 * m - 2)
884                 {
885                     triangulized = true;
886                 }
887                 if (A[i][i] == 0 && i < 2 * m - 2) {
888                     // No s'ha trobat una fila no nul.la per intercanviar. Sortir amb error.
889                     printf("No es pot triangulitzar la matriu, s'ha trobat un pivot de zero!\n");
890                     return;
891                 }
892 
893             }
894             if (!triangulized)
895             {
896                 for (k = i + 1; k < 2 * m; k++) {
897                     c = A[k][i] / A[i][i];
898                     for (j = i; j < 2 * m; j++) {
899                         A[k][j] -= c * A[i][j];
900                     }
901                 }
902             }
903         }
904 
905         aux = backSubstitution(A, 2 * m);
906 
907         j = 0;
908         for (i = 0; i < 2 * m; i++) {
909             if (i % 2 == 0) {
910                 vep_re[j] = aux[i];
911             }
912             else {
913                 vep_im[j] = aux[i];
914                 j++;
915             }
916         }
917 
918         assignar_vect_a_col(VEPs, vep_re, 2, 4);
919         assignar_vect_a_col(VEPs, vep_im, 3, 4);
920         free(aux);
921         alliberarMat(A, 2 * m);
922     }
923 }
924 
925 void Ktheta(real** KG, real* theta, int N, real** VEPs, real* Li, int col1, int col2)
926 {
927     {
928         real* u_p = crearVect(DIM);
```

```
929         real* v_p = crearVect(DIM);
930 
931         for (int i = 0; i < DIM; i++) //pq son DIM files
932         {
933             u_p[i] = VEPs[i][col1];  //col1=columna q volem.
934             v_p[i] = VEPs[i][col2];
935         }
936 
937         for (int i = 0; i < DIM; i++)
938         {
939             for (int j = 0; j < N; j++)
940             {
941                 KG[i][j] = Li[i] + (u_p[i] * cosl(pi2 * theta[j]) - v_p[i] *
     sinl(pi2 * theta[j])) * R;
942             }
943         }
944         //Alliberem memoria
945         free(u_p);
946         free(v_p);
947     }
948 }
949 
950 void camp_sistema_en_grid(real** KG, real** XKG, real** DG, int N)
951 {
952     {
953         real    x, y, z;
954         real    px, py, pz;
955         real    U_x, U_y, U_z; //Primeres derivades de U
956         real    U_xx, U_xy, U_yx, U_xz, U_zx, U_yy, U_yz, U_zy, U_zz;   //
    Segones derivades de U
957         real    r12;    //r_1^2
958         real    r22;    //r_2^2
959         real    r13;    //r_1^3=r_1^2*r_1
960         real    r23;    //r_2^3=r_2^2*r_2
961         real    r15;    //r_1^5=r_1^3*r_1^2
962         real    r25;    //r_2^5=r_2^3*r_2^2
963 
964         real** DX_peque = crearMat(DIM, N);
965 
966         int DIMxi;
967 
968         for (int i = 0; i < N; i++)
969         {
970             DIMxi = i * DIM;
971             x = KG[0][i];
972             px = KG[1][i];
973             y = KG[2][i];
974             py = KG[3][i];
975             z = KG[4][i];
976             pz = KG[5][i];
977 
978             r12 = (x + MU) * (x + MU) + y * y + z * z;;//r_1^2
979             r22 = (x - (1.0L - MU)) * (x - (1.0L - MU)) + y * y + z * z;//
    r_2^2
980 
981             r13 = r12 * sqrtl(r12);      //r_1^3=r_1^2*r_1
982             r23 = r22 * sqrtl(r22);      //r_2^3=r_2^2*r_2
983 
984             //Primeres derivades de U:
```

```
985             U_x = -x + ((1.0L - MU) * (x + MU)) / r13 + (MU * (x - (1.0L -
    MU))) / r23;
986             U_y = (-1.0L + (1.0L - MU) / r13 + MU / r23) * y;
987             U_z = ((1.0L - MU) / r13 + MU / r23) * z;
988 
989             //Camp
990             XKG[0][i] = px + y;
991             XKG[1][i] = py - x - U_x;   //py-x-U_x
992             XKG[2][i] = py - x;
993             XKG[3][i] = -px - y - U_y; //-px-y-U_y
994             XKG[4][i] = pz;
995             XKG[5][i] = -U_z;           //-U_z
996 
997             r15 = r13 * r12;      //r_1^5=r_1^3*r_1^2
998             r25 = r23 * r22;      //r_2^5=r_2^3*r_2^2
999 
1000            //Segones derivades de U:
1001            U_xx = -1.0L + (1.0L - MU) / r13 + MU / r23 - 3.0L * ((1.0L - MU
    ) * (x + MU) * (x + MU)) / r15 - 3.0L * MU * ((x - (1.0L - MU)) * (x -
    (1.0L - MU))) / r25;
1002            U_xy = U_yx = -3.0L * y * (((1 - MU) * (x + MU)) / r15 + (MU * (
    x - (1.0L - MU))) / r25);
1003            U_xz = U_zx = -3.0L * z * (((1 - MU) * (x + MU)) / r15 + (MU * (
    x - (1.0L - MU))) / r25);
1004            U_yy = -1.0L + (1.0L - MU) / r13 + MU / r23 - 3.0L * y * y *
    ((1.0L - MU) / r15 + MU / r25);
1005            U_yz = U_zy = -3.0L * y * z * ((1.0L - MU) / r15 + MU / r25);
1006            U_zz = (1.0L - MU) / r13 + MU / r23 - 3.0L * z * z * ((1.0L - MU
    ) / r15 + MU / r25);
1007 
1008            DX_peque[0][1] = 1.0L;
1009            DX_peque[0][2] = 1.0L;
1010 
1011            DX_peque[1][0] = -1.0L - U_xx;
1012            DX_peque[1][2] = -U_xy;
1013            DX_peque[1][3] = 1.0L;
1014            DX_peque[1][4] = -U_xz;
1015 
1016            DX_peque[2][0] = -1.0L;
1017            DX_peque[2][3] = 1.0L;
1018 
1019            DX_peque[3][0] = -U_yx;
1020            DX_peque[3][1] = -1.0L;
1021            DX_peque[3][2] = -1.0L - U_yy;
1022            DX_peque[3][4] = -U_yz;
1023 
1024            DX_peque[4][5] = 1.0L;
1025 
1026            DX_peque[5][0] = -U_zx;
1027            DX_peque[5][2] = -U_zy;
1028            DX_peque[5][4] = -U_zz;
1029 
1030            for (int k = 0; k < DIM; k++)
1031            {
1032                for (int j = 0; j < DIM; j++)
1033                {
1034                    DG[DIMxi + k][DIMxi + j] = DX_peque[k][j];
1035                }
1036            }
```

```
1037         }
1038         alliberarMat(DX_peque, DIM);
1039     }
1040 }
1041
1042 void four1(real* data, int nn, int isign)
1043 {
1044     int n, mmax, m, j, istep, i, aux_i_re, aux_i_co, aux_j_re, aux_j_co;
1045     real wtemp, wr, wpr, wpi, wi, theta;
1046     real tempr, tempi;
1047     real val0, val1, val2;
1048
1049     val0 = (real)0;
1050     val1 = (real)1;
1051     val2 = (real)2;
1052
1053     val0 = 0.0L;
1054     val1 = 1.0L;
1055     val2 = 2.0L;
1056
1057     n = nn << 1; //n = nn * 2;
1058     j = 1;
1059
1060     for (i = 1; i < n; i += 2)
1061     {
1062         //Posem els indexs q toquen per com tenim ordenat el nostre:
1063         aux_i_re = num_com_toca_ordre_meu(i, nn);
1064         aux_j_re = num_com_toca_ordre_meu(j, nn);
1065
1066         if (j > i)//Bit-reversal section of the routine. Exchange the two
    complex numbers.
1067         {
1068
1069             aux_i_re = num_com_toca_ordre_meu(i, nn);
1070             aux_j_re = num_com_toca_ordre_meu(j, nn);
1071             aux_i_co = num_com_toca_ordre_meu(i + 1, nn);
1072             aux_j_co = num_com_toca_ordre_meu(j + 1, nn);
1073
1074             SWAP(data[aux_j_re], data[aux_i_re]);
1075             SWAP(data[aux_i_co], data[aux_j_co]);
1076         }
1077         m = n >> 1;
1078         while (m >= 2 && j > m)
1079         {
1080             j -= m;
1081             m >>= 1;     // m / = 2;
1082         }
1083         j += m;
1084     }
1085
1086     //Here begins Danielson-Lanczos section of the routine.
1087     mmax = 2;
1088     while (n > mmax)//Outer loop executed log_2 (nn) times.
1089     {
1090         istep = mmax << 1;
1091         theta = isign * (pi2 / mmax);
1092         wtemp = sinl(theta / val2);
1093         wpr = val0 - val2 * wtemp * wtemp;
1094         wpi = sinl(theta);
```

```
1095        wr = val1;
1096        wi = val0;
1097        for (m = 1; m < mmax; m += 2)
1098        {
1099            for (i = m; i <= n; i += istep)
1100            {
1101                j = i + mmax;
1102                aux_i_re = num_com_toca_ordre_meu(i, nn);
1103                aux_j_re = num_com_toca_ordre_meu(j, nn);
1104                aux_i_co = num_com_toca_ordre_meu(i + 1, nn);
1105                aux_j_co = num_com_toca_ordre_meu(j + 1, nn);
1106
1107
1108                tempr = wr * data[aux_j_re] - wi * data[aux_j_co];
1109                tempi = wr * data[aux_j_co] + wi * data[aux_j_re];
1110                data[aux_j_re] = data[aux_i_re] - tempr;
1111                data[aux_j_co] = data[aux_i_co] - tempi;
1112                data[aux_i_re] += tempr;
1113                data[aux_i_co] += tempi;
1114
1115            }
1116            //Trigonometris recurrence
1117            wtemp = wr;
1118            wr = wtemp * wpr - wi * wpi + wr;
1119            wi = wi * wpr + wtemp * wpi + wi;
1120        }
1121        mmax = istep;
1122    }
1123    return;
1124 }
1125
1126 void iderivativeF(real* data, int nn)
1127
1128 {
1129     real dfreal, dfimag;
1130     int  aux_k_co;
1131     int  aux_k_re;
1132
1133     for (int k = 0; k < nn / 2; k++) {
1134         aux_k_co = num_com_toca_deriv(2 * k + 1, nn);
1135         aux_k_re = num_com_toca_deriv(2 * k, nn);
1136         dfreal = -pi2 * k * data[aux_k_co];
1137         dfimag = pi2 * k * data[aux_k_re];
1138         data[aux_k_re] = dfreal;
1139         data[aux_k_co] = dfimag;
1140     }
1141     for (int k = nn / 2; k < nn; k++) {
1142         aux_k_co = num_com_toca_deriv(2 * k + 1, nn);
1143         aux_k_re = num_com_toca_deriv(2 * k, nn);
1144         dfreal = -pi2 * (k - nn) * data[aux_k_co];
1145         dfimag = pi2 * (k - nn) * data[aux_k_re];
1146         data[aux_k_re] = dfreal;
1147         data[aux_k_co] = dfimag;
1148     }
1149 }
1150
1151 void iG2F(real* data, int nn)
1152 {
1153     four1(data, nn, -1);
```

```
1154     for (int i = 0; i < nn; i++)
1155     {
1156         data[i] /= nn;
1157         data[nn + i] /= nn;
1158 
1159     }
1160 
1161 }
1162 
1163 void iF2G(real* data, int nn)
1164 {
1165     four1(data, nn, 1);
1166 }
1167 
1168 int newton_bestia(real** KG, real* wp, real h, real** D_m, int N, real** DG)
1169 {
1170 
1171     int result;
1172     {
1173         int      jxDIM;
1174         int      N_2 = 2 * N;
1175         int      NxDIM = DIM * N;
1176         int      fil_DG = NxDIM + 2;
1177         int      col_DG = NxDIM + 1;
1178         int      compt = 0;
1179 
1180         real** KF;       // =FFT(KG)
1181         real** DKF;      // DKF es la derivada de KF.
1182         real** DK;       // Derivada amb fourier de KG.
1183         real** XKG;      // Matriu del camp en els punts de la grid.
1184 
1185         real* K0;
1186         real* DHK_0;    //Dif HK_0
1187         real* Dg;
1188         real* increments;
1189         real* G;        //Errors
1190         real* G_neg;
1191 
1192         real      e_g; //error seccio transversal
1193         real      e_H; //error energia
1194         real      val_max = 0.0L;
1195         real      HK_0;
1196         real      max_G;
1197 
1198         for (int i = 0; i < fil_DG; i++)
1199             for (int j = 0; j < col_DG; j++)
1200                 DG[i][j] = 0.0L;
1201 
1202         //Creem la matriu q necessitarem guardar per fer el refinament.
1203         do
1204         {
1205             compt++;
1206             ////////////////////////////////
1207             //       Omplim G i DG        ///
1208             ////////////////////////////////
1209 
1210             XKG = crearMat(DIM, N);
1211 
1212             camp_sistema_en_grid(KG, XKG, DG, N);
```

```
1213
1214            //
1215            //Calculem la derivada de KG, DK amb fourier.
1216            //
1217
1218            //Reservem memoria
1219            KF = crearMat(DIM, N_2);
1220            DKF = crearMat(DIM, N_2);
1221            DK = crearMat(DIM, N_2);
1222            for (int i = 0; i < DIM; i++)
1223            {
1224                memcpy(KF[i], KG[i], N_2 * sizeof(real));
1225                iG2F(KF[i], N);             // KF = FFT(KG).
1226                printf("Nyquist!=% Le\t", KF[i][N / 2]);
1227                printf("Nyquist2!=% Le\n", KF[i][N / 2 + N]);
1228                KF[i][N / 2 + N] = 0.0L;//Hem cnviat!
1229                KF[i][N / 2] = 0.0L;//igualem a 0 el terme Nyquist.
1230                memcpy(KG[i], KF[i], N_2 * sizeof(real));
1231                iF2G(KG[i], N);
1232
1233                memcpy(DKF[i], KF[i], N_2 * sizeof(real));
1234                iderivativeF(DKF[i], N);    // DKF = derivada(KF).
1235                memcpy(DK[i], DKF[i], N_2 * sizeof(real));
1236                iF2G(DK[i], N);             // DK = IFFT(DKF).
1237            }
1238            alliberarMat(KF, DIM);
1239            alliberarMat(DKF, DIM);
1240
1241
1242            printf("comprovem l'error de Dm\n");
1243            real* KGvec = crearVect(NxDIM);
1244            real* DKvec = crearVect(NxDIM);
1245
1246            for (int j = 0; j < N; j++)
1247                for (int i = 0; i < DIM; i++)
1248                {
1249                    KGvec[i + j * DIM] = KG[i][j];
1250                    DKvec[i + j * DIM] = DK[i][j];
1251                }
1252
1253            real* vector_mult = mult_mat_vect(D_m, KGvec, NxDIM, NxDIM);
1254
1255            for (int i = 0; i < NxDIM; i++)
1256                vector_mult[i] -= DKvec[i];
1257
1258            printf("El max de l'error de DM ess: % .Le", max_vec_abs(
    vector_mult, NxDIM));
1259
1260            free(vector_mult);
1261            free(KGvec);
1262            free(DKvec);
1263
1264            //Comprovem l'error de la derivada comparant amb diferencies
    finites.
1265            {
1266                real* error_derivada = crearVect(N);
1267                for (int i = 1; i < (N - 1); i++)
1268                {
1269                    error_derivada[i - 1] = DK[1][i] - (KG[1][i + 1] - KG
```

```
    [1][i - 1]) / (2.0L / N);
1270            }
1271
1272            printf("N=%d\n", N);
1273            printf("error maxim d la derivada DK, comparant-la amb
    diferencies finites es: % Le\n", max_vec_abs(error_derivada, N));
1274            free(error_derivada);
1275        }
1276
1277        //Omplim l'ultima columna de DG
1278        jxDIM = 0;
1279        for (int j = 0; j < N; j++)
1280        {
1281            jxDIM = j * DIM;
1282            for (int i = 0; i < DIM; i++)
1283                DG[jxDIM + i][col_DG - 1] = -DK[i][j];
1284        }
1285
1286        real* XK0 = crearVect(DIM);
1287
1288        //eN AQUEST CAS FEM SERVIR K0=H0
1289        for (int i = 0; i < DIM; i++) {
1290            XK0[i] = XKG[i][0];
1291        }
1292
1293        DHK_0 = deriv_hamiltonia_en_camp(XK0);//Dh_K0
1294        Dg = proj_eix_k(cy, DIM);//=Dg_K0. Per orbites planes, cy. Per
    orbites verticals, es posara cz. TODO.
1295
1296        //Omplim les ultimes dues files de DG amb els valors
    corresponents.
1297        for (int i = 0; i < DIM; i++)
1298        {
1299            DG[fil_DG - 2][i] = DHK_0[i];
1300            DG[fil_DG - 1][i] = Dg[i];
1301        }
1302        free(DHK_0);
1303        free(Dg);
1304
1305        //Test. Quina seria la frequencia ideal?
1306        //real numerador = 0.0L, denominador = 0.0L;
1307        //for (int j = 0; j < N; j++)
1308        //    for (int i = 0; i < DIM; i++) {
1309        //        numerador += XKG[i][j] * DK[i][j];
1310        //        denominador += DK[i][j] * DK[i][j];
1311        //    }
1312        //printf("wp abans del nou me: % .12Le\n", *wp);
1313        //*wp = numerador / denominador;
1314        //printf("wp dspss del nou me: % .12Le\n", *wp);
1315
1316        //DG=DG-wD_m.
1317        for (int i = 0; i < NxDIM; i++)
1318            for (int j = 0; j < NxDIM; j++)
1319                DG[i][j] -= (*wp) * D_m[i][j];
1320        printf("%.8Le\n", max_mat_abs(DG, fil_DG, col_DG));
1321        //Ara ja tenim plena la matriu DG!
1322
1323        //
1324        //Omplim G
```

```
1325            //
1326
1327            G = crearVect(fil_DG);
1328
1329            printf("max DK=%.8Le\n", max_mat_abs(DK, DIM, N_2));
1330            printf("max XKG=%.8Le\n", max_mat_abs(XKG, DIM, N));
1331            //Omplim les primeres files de G (nomes en faltaran dues: e_h i
    e_g)
1332            for (int j = 0; j < N; j++)
1333            {
1334                jxDIM = DIM * j;
1335                for (int i = 0; i < DIM; i++)
1336                    G[i + jxDIM] = XKG[i][j] - DK[i][j] * (*wp);
1337            }
1338            printf("max G=%.8Le\n", max_vec_abs(G, fil_DG));
1339            free(DK);
1340            free(XKG);
1341            K0 = crearVect(DIM);
1342            for (int i = 0; i < DIM; i++) {
1343                K0[i] = KG[i][0];
1344            }
1345
1346            HK_0 = hamiltonia(K0);
1347
1348            printf("N=%d\t r=% Le\n", N, R);
1349            printf("L'errror max de X(K(theta_i))-DK(theta_i)w \'es :%Le\n",
     max_vec_abs(G, fil_DG));
1350            e_H = HK_0 - h;//Error del hamilt.
1351
1352            e_g = K0[cy];//Error de la seccio transversal
1353            free(K0);
1354            G[NxDIM] = e_H;
1355            G[NxDIM + 1] = e_g;
1356            printf("e_H=%Le\n", e_H);
1357            printf("e_g=%Le\n", e_g);
1358            printf("L'error maxim de G es: % .8Le\n", max_vec_abs(G, fil_DG)
    );
1359
1360            //Ja tenim G
1361            max_G = max_vec_abs(G, fil_DG);
1362            if (max_G > tol_newton || compt == 1)//ha d'haver entrarty
    mÃnim una vegada!
1363            {
1364
1365
1366                ////////////////////////////////////////////////////
1367                //          Resolem el sistema lineal amb SVD         //
1368                ////////////////////////////////////////////////////
1369
1370                G_neg = G;
1371                for (int i = 0; i < fil_DG; i++)
1372                    G_neg[i] = -G[i];
1373
1374                //Resolem DG(K,w)*(deltaK ,w)=-G(K,w)
1375
1376                increments = resol_sist_lin_amb_SVD(DG, G_neg, fil_DG,
    col_DG);
1377                free(G);//tb es G_neg
```

```
1378                 printf("correcio w: % .8Le", increments[col_DG - 1]);
1379                 //Ja tenim la solucio, es increments.
1380 
1381                 val_max = max_vec_abs(increments, col_DG);
1382                 printf("\nEl valor maxim de l'increment en el proced de
     Newton num %d es: %Le\n", compt, val_max);
1383 
1384                 //Ja l'hem resolt.
1385 
1386                 //Ara s'ha de tornar a fer tot el procediment per una nova
     KG i la nova w
1387 
1388                 ////////////////////////////////
1389                 //      Actualitzem  KG i w    //
1390                 ////////////////////////////////
1391 
1392                 for (int j = 0; j < N; j++)
1393                 {
1394                     jxDIM = j * DIM;
1395                     for (int i = 0; i < DIM; i++)
1396                         KG[i][j] += increments[jxDIM + i];
1397                 }
1398 
1399                 //Actualitzem  w.
1400 
1401                 printf("wp durant el Newton, abans d'actualitzar-la=%.8Le\n"
     , *wp);
1402                 *wp = *wp + increments[NxDIM];
1403                 printf("wp durant el Newton, abans despres d'actualitzar-la
     =%.8Le\n", *wp);
1404                 printf("%.8Le\n", max_mat_abs(KG, DIM, N_2));
1405                 free(increments);
1406             }
1407 
1408         } while (compt <= 4 && max_G > tol_newton);
1409 
1410         if (max_G < tol_newton)
1411         {
1412             //DG ja es la q toca
1413             result = compt;//ha convergit en compt vegades
1414         }
1415         else
1416         {
1417             result = 0;//no ha convergit
1418         }
1419     }
1420 
1421     return result;
1422 }
1423 
1424 int nextPowerOfTwo(int n) {
1425     int PowerOfTwo = 1;
1426     {
1427         if (n <= 0) {
1428             return 1; // Tornar 1 per nums negatius.
1429         }
1430 
1431         while (PowerOfTwo < n) {
1432             PowerOfTwo <<= 1;  // Equivalent a multiplicar por 2
```

```
1433         }
1434     }
1435     return PowerOfTwo;
1436 }
1437 
1438 int svdcmp(real** a, int nRows, int nCols, real* w, real** v) {
1439     int flag, i, its, j, jj, k, l, nm;
1440     real anorm, c, f, g, h, s, scale, x, y, z, *rv1;
1441 
1442     rv1 = (real*)malloc(sizeof(real) * nCols);
1443     if (rv1 == NULL) {
1444         printf("svdcmp(): Unable to allocate vector\n");
1445         return (-1);
1446     }
1447 
1448     g = scale = anorm = 0.0L;
1449     for (i = 0; i < nCols; i++) {
1450         l = i + 1;
1451         rv1[i] = scale * g;
1452         g = s = scale = 0.0L;
1453         if (i < nRows) {
1454             for (k = i; k < nRows; k++)
1455                 scale += fabsl(a[k][i]);
1456             if (scale) {
1457                 for (k = i; k < nRows; k++) {
1458                     a[k][i] /= scale;
1459                     s += a[k][i] * a[k][i];
1460                 }
1461                 f = a[i][i];
1462                 g = -SIGN(sqrtl(s), f);
1463                 h = f * g - s;
1464                 a[i][i] = f - g;
1465                 for (j = l; j < nCols; j++) {
1466                     for (s = 0.0L, k = i; k < nRows; k++)
1467                         s += a[k][i] * a[k][j];
1468                     f = s / h;
1469                     for (k = i; k < nRows; k++)
1470                         a[k][j] += f * a[k][i];
1471                 }
1472                 for (k = i; k < nRows; k++)
1473                     a[k][i] *= scale;
1474             }
1475         }
1476         w[i] = scale * g;
1477         g = s = scale = 0.0L;
1478         if (i < nRows && i != nCols - 1) {
1479             for (k = l; k < nCols; k++)
1480                 scale += fabsl(a[i][k]);
1481             if (scale) {
1482                 for (k = l; k < nCols; k++) {
1483                     a[i][k] /= scale;
1484                     s += a[i][k] * a[i][k];
1485                 }
1486                 f = a[i][l];
1487                 g = -SIGN(sqrtl(s), f);
1488                 h = f * g - s;
1489                 a[i][l] = f - g;
1490                 for (k = l; k < nCols; k++)
1491                     rv1[k] = a[i][k] / h;
```

```
1492                 for (j = l; j < nRows; j++) {
1493                     for (s = 0.0L, k = l; k < nCols; k++)
1494                         s += a[j][k] * a[i][k];
1495                     for (k = l; k < nCols; k++)
1496                         a[j][k] += s * rv1[k];
1497                 }
1498                 for (k = l; k < nCols; k++)
1499                     a[i][k] *= scale;
1500             }
1501         }
1502         anorm = FMAX(anorm, (fabsl(w[i]) + fabsl(rv1[i])));
1503 
1504         printf(".");
1505         fflush(stdout);
1506     }
1507 
1508     for (i = nCols - 1; i >= 0; i--) {
1509         if (i < nCols - 1) {
1510             if (g) {
1511                 for (j = l; j < nCols; j++)
1512                     v[j][i] = (a[i][j] / a[i][l]) / g;
1513                 for (j = l; j < nCols; j++) {
1514                     for (s = 0.0L, k = l; k < nCols; k++)
1515                         s += a[i][k] * v[k][j];
1516                     for (k = l; k < nCols; k++)
1517                         v[k][j] += s * v[k][i];
1518                 }
1519             }
1520             for (j = l; j < nCols; j++)
1521                 v[i][j] = v[j][i] = 0.0L;
1522         }
1523         v[i][i] = 1.0L;
1524         g = rv1[i];
1525         l = i;
1526         printf(".");
1527         fflush(stdout);
1528     }
1529 
1530     for (i = IMIN(nRows, nCols) - 1; i >= 0; i--) {
1531         l = i + 1;
1532         g = w[i];
1533         for (j = l; j < nCols; j++)
1534             a[i][j] = 0.0L;
1535         if (g) {
1536             g = 1.0L / g;
1537             for (j = l; j < nCols; j++) {
1538                 for (s = 0.0L, k = l; k < nRows; k++)
1539                     s += a[k][i] * a[k][j];
1540                 f = (s / a[i][i]) * g;
1541                 for (k = i; k < nRows; k++)
1542                     a[k][j] += f * a[k][i];
1543             }
1544             for (j = i; j < nRows; j++)
1545                 a[j][i] *= g;
1546         }
1547         else
1548             for (j = i; j < nRows; j++)
1549                 a[j][i] = 0.0L;
1550         ++a[i][i];
```

```
1551        printf(".");
1552        fflush(stdout);
1553    }
1554
1555    for (k = nCols - 1; k >= 0; k--) {
1556        for (its = 0; its < 30; its++) {
1557            flag = 1;
1558            for (l = k; l >= 0; l--) {
1559                nm = l - 1;
1560                if ((fabs(rv1[l]) + anorm) == anorm) {
1561                    flag = 0;
1562                    break;
1563                }
1564                if ((fabs(w[nm]) + anorm) == anorm)
1565                    break;
1566            }
1567            if (flag) {
1568                c = 0.0L;
1569                s = 1.0L;
1570                for (i = l; i <= k; i++) {
1571                    f = s * rv1[i];
1572                    rv1[i] = c * rv1[i];
1573                    if ((fabs(f) + anorm) == anorm)
1574                        break;
1575                    g = w[i];
1576                    h = pythag(f, g);
1577                    w[i] = h;
1578                    h = 1.0L / h;
1579                    c = g * h;
1580                    s = -f * h;
1581                    for (j = 0; j < nRows; j++) {
1582                        y = a[j][nm];
1583                        z = a[j][i];
1584                        a[j][nm] = y * c + z * s;
1585                        a[j][i] = z * c - y * s;
1586                    }
1587                }
1588            }
1589            z = w[k];
1590            if (l == k) {
1591                if (z < 0.0L) {
1592                    w[k] = -z;
1593                    for (j = 0; j < nCols; j++)
1594                        v[j][k] = -v[j][k];
1595                }
1596                break;
1597            }
1598            if (its == 29)
1599                printf("en 30 iteracions svdcmp no convergeix\n");
1600            x = w[l];
1601            nm = k - 1;
1602            y = w[nm];
1603            g = rv1[nm];
1604            h = rv1[k];
1605            f = ((y - z) * (y + z) + (g - h) * (g + h)) / (2.0L * h * y);
1606            g = pythag(f, 1.0L);
1607            f = ((x - z) * (x + z) + h * ((y / (f + SIGN(g, f))) - h)) / x;
1608            c = s = 1.0L;
1609            for (j = l; j <= nm; j++) {
```

```
1610                 i = j + 1;
1611                 g = rv1[i];
1612                 y = w[i];
1613                 h = s * g;
1614                 g = c * g;
1615                 z = pythag(f, h);
1616                 rv1[j] = z;
1617                 c = f / z;
1618                 s = h / z;
1619                 f = x * c + g * s;
1620                 g = g * c - x * s;
1621                 h = y * s;
1622                 y *= c;
1623                 for (jj = 0; jj < nCols; jj++) {
1624                     x = v[jj][j];
1625                     z = v[jj][i];
1626                     v[jj][j] = x * c + z * s;
1627                     v[jj][i] = z * c - x * s;
1628                 }
1629                 z = pythag(f, h);
1630                 w[j] = z;
1631                 if (z) {
1632                     z = 1.0L / z;
1633                     c = f * z;
1634                     s = h * z;
1635                 }
1636                 f = c * g + s * y;
1637                 x = c * y - s * g;
1638                 for (jj = 0; jj < nRows; jj++) {
1639                     y = a[jj][j];
1640                     z = a[jj][i];
1641                     a[jj][j] = y * c + z * s;
1642                     a[jj][i] = z * c - y * s;
1643                 }
1644             }
1645             rv1[l] = 0.0L;
1646             rv1[k] = f;
1647             w[k] = x;
1648         }
1649         printf(".");
1650         fflush(stdout);
1651     }
1652     printf("\n");
1653 
1654     free(rv1);
1655 
1656     return (0);
1657 }
1658 
1659 int num_com_toca_ordre_meu(int num, int nn)
1660 {
1661     int aux;
1662     {
1663         if (num % 2 != 0) {
1664             aux = (num - 1) / 2;
1665         }
1666         else {
1667             aux = nn - 1 + num / 2;
1668         }
```

```
1669     }
1670     return aux;
1671 }
1672 
1673 int num_com_toca_deriv(int num, int nn)
1674 {
1675     int aux;
1676     {
1677         if (num % 2 == 0) {
1678             aux = (num) / 2;
1679         }
1680         else {
1681             aux = nn - 1 + (num + 1) / 2;
1682         }
1683     }
1684     return aux;
1685 }
1686 
1687 void provatura_hamilt_dif(void)
1688 {
1689     {
1690         int N = 128;
1691         real beta = 1.0L;
1692         real w = beta / pi2;
1693         //ham = -1.0L / 2.0L * beta * (x * x + y * y) - x * x * x / 3.0L;
1694         real x, y;
1695         real r = 1.0E-5;
1696 
1697         real* theta = crearVect(N);
1698         real** KG = crearMat(2, N * 2);
1699         real** XKG = crearMat(2, N);
1700         real** KF = crearMat(2, N*2);        //=FFT(KG). Despres haurem de
    derivar i fer la IFFT(D(KF)).
1701         real** DKF = crearMat(2, N*2);       //DKF es la derivada de KF.
1702         real** DK = crearMat(2, N*2);        //DK es la IFFT de la derivada
    de la FFT de KG.i.e. la derivada amb fourier de KG.
1703 
1704         theta = Theta(N);
1705         for (int j = 0; j < N; j++)
1706         {
1707             for (int i = 0; i < 2; i++)
1708             {
1709                 if(i==0)
1710                 {
1711                     KG[i][j] = r * cosl(pi2 * theta[j]);
1712                 }
1713                 else if (i == 1)
1714                 {
1715                     KG[i][j] = r * sinl(pi2 * theta[j]);
1716                 }
1717             }
1718         }
1719         //camp.
1720         for (int i = 0; i < N; i++)
1721         {
1722             x = KG[0][i];
1723             y = KG[1][i];
1724             XKG[0][i] = -beta * y;
1725             XKG[1][i] = beta * x + x * x;
```

```
1726        }
1727        //Calculem la derivada de KG. A saco, amb fourier.
1728        for (int i = 0; i < 2; i++)
1729        {
1730            memcpy(KF[i], KG[i], (N * 2) * sizeof(real));
1731            iG2F(KF[i], N);              // KF = FFT(KG).
1732            printf("Nyquist!=% Le\n", KF[i][N / 2]);
1733            KF[i][N / 2]=0.0L;
1734            memcpy(DKF[i], KF[i], (N * 2) * sizeof(real));
1735            iderivativeF(DKF[i], N);    // DKF = derivada(KF).
1736            memcpy(DK[i], DKF[i], (N * 2) * sizeof(real));
1737            iF2G(DK[i], N);              // DK = IFFT(DKF).
1738        }
1739        real** DK_w;
1740        DK_w = mult_mat_escal(DK, w, 2, N);
1741        real* ERROR = crearVect(2* N + 2);
1742        for (int k = 0; k < N; k++)
1743            for (int i = 0; i < 2; i++)
1744                ERROR[i + 2 * k] = XKG[i][k] - DK_w[i][k];
1745        FILE* arxiu_err = NULL;
1746        imprimir_vector_en_dues_col_arxiu(ERROR, 2 * N, arxiu_err);
1747        alliberarMat(DK_w, 2);
1748
1749    }
1750 }
```

Codi 4: Definició de funcions

```
1    //Funcions generiques
2
3  real** mult_mat_mat(real** matriu1, real** matriu2, int files1, int col1,
       int files2, int col2)
4  {
5      real** resultat;
6      {
7          if (col1 != files2)
8          {
9              printf("Estem intentant multiplicar una matriu que no es pot
       multiplicar.\n");
10             exit(2);
11         }
12         else
13         {
14             resultat = crearMat(files1, col2);
15             real suma;
16             for (int i = 0; i < files1; i++)
17                 for (int j = 0; j < col2; j++) {
18                     suma = 0.0L;
19                     for (int k = 0; k < col1; k++)
20                         suma += matriu1[i][k] * matriu2[k][j];
21                     resultat[i][j] = suma;
22                 }
23         }
24     }
25     return resultat;
26 }
27
28 real** mult_mat_escal(real** mat, real escal, int files, int col)
29 { //multiplicar matriu i escalar.
```

```
30      real** resultat;
31      {
32          resultat = crearMat(files, col);
33          for (int i = 0; i < files; i++) {
34              for (int j = 0; j < col; j++)
35                  resultat[i][j] = mat[i][j] * escal;
36
37          }
38      }
39      return resultat;
40  }
41
42  real** restar_mat_mat(real** matriu1, real** matriu2, int fil1, int col1,
        int fil2, int col2)
43  {   //Restar dues matrius
44      if (fil1 != fil2 || col1 != col2) {
45          printf("aquesta resta de matrius no es possible pq files i columnes
        han de coincidir per matrius");
46          exit(2);
47      }
48      real** resultat;
49      {
50          resultat = crearMat(fil1, col1);
51          for (int i = 0; i < fil1; i++) {
52              for (int j = 0; j < col1; j++) {
53                  resultat[i][j] = matriu1[i][j] - matriu2[i][j];
54              }
55          }
56      }
57      return resultat;
58  }
59
60  real** crearMat(int files, int columnes)
61  {
62      real** matriu;
63      {
64
65          // Reserva memoria per a les files
66          matriu = (real**)calloc(files, sizeof(real*));
67          if (matriu == NULL)
68          {
69              printf("Error a l'assignar memoria per a les files.\n");
70              return NULL;
71          }
72          // Reserva memoria per a cada columna en cada fila
73          for (int i = 0; i < files; i++)
74          {
75              matriu[i] = (real*)calloc(columnes, sizeof(real));
76              if (matriu[i] == NULL)
77              {
78                  printf("Error a l'assignar memoria per a les columnes.\n");
79                  return NULL;
80              }
81              for (int j = 0; j < columnes; j++)
82                  matriu[i][j] = 0.0L;
83          }
84      }
85      return matriu;
86  }
```

```
87
88 real** crear_mat_identitat(int files, int columnes)
89 {
90     real** matriu;
91     {
92         if (files != columnes) {
93             printf("Intentant crear una matriu identitat NO quadrada!\n");
94             exit(1);
95         }
96         matriu = crearMat(files, columnes);
97         for (int i = 0; i < files; i++)
98             matriu[i][i] = 1.0L;
99     }
100    return matriu;
101 }
102
103
104 real* mult_mat_vect(real** matriu, real* vect, int files, int columnes)
105 {//multiplicar matriu i vector.
106     real* resultat;
107     {
108         resultat = crearVect(files);
109         for (int i = 0; i < files; i++) {
110             resultat[i] = 0.0L;
111             for (int j = 0; j < columnes; j++) {
112                 resultat[i] += matriu[i][j] * vect[j];
113             }
114             //Eliminem errors numerics
115             if (fabsl(resultat[i]) < tol_zero)
116                 resultat[i] = 0.0L;
117         }
118     }
119     return resultat;
120 }
121
122 real* mult_escal_vect(real escal, real* vect, int files)
123 { //multiplicar escalar i vector.
124     real* resultat;
125     {
126         resultat = crearVect(files);
127         for (int i = 0; i < files; i++) {
128             resultat[i] = vect[i] * escal;
129
130         }
131     }
132     return resultat;
133 }
134
135 real* crearVect(int n)
136 {
137     real* vec;
138     {
139         vec = (real*)calloc(n, sizeof(real));
140         if (vec == NULL)
141         {
142             printf("Error a l'assignar memoria en el vector\n");
143             return NULL;
144         }
145         for (int i = 0; i < n; i++)
```

```
146             vec[i] = 0.0L;
147     }
148     return vec;
149 
150 }
151 
152 
153 real* restar_vect(real* v1, real* v2, int files)
154 {   //Restar dos vectors
155     real* resultat;
156     {
157         resultat = crearVect(files);
158         for (int i = 0; i < files; i++) {
159             resultat[i] = v1[i] - v2[i];
160             if (resultat[i] < 1E-20)
161                 resultat[i] = 0.0L;
162         }
163     }
164     return resultat;
165 }
166 
167 real mult_dos_vect_a_real(real* v1, real* v2, int n)
168 {
169     real resultat;
170     {
171         resultat = 0.0L;
172         for (int i = 0; i < n; i++)
173             resultat += v1[i] * v2[i];
174     }
175     return resultat;
176 }
177 
178 real max_vec_abs(real *vec, int n) {
179     real max;
180     {
181         real abs_val;
182 
183         max = fabsl(vec[0]);
184         for (int i = 1; i < n; i++) {
185             abs_val = fabsl(vec[i]); // Calcula el valor absolut de l'
    element actual.
186             if (abs_val > max) {
187                 max = abs_val; // Actualiza el mÃ x, si cal.
188             }
189         }
190     }
191     return max;
192 }
193 
194 real max_mat_abs(real **vec, int fil,int col) {
195     real max;
196     {
197         real abs_val;
198         max = fabsl(vec[0][0]);
199         for (int i = 0; i < fil; i++) {
200             for(int j=0;j<col;j++)
201             {
202                 abs_val = fabsl(vec[i][j]); // Calcula el valor absolut de l
```

```
    'element actual.
203                 if (abs_val > max) {
204                     max = abs_val; // Actualiza el max, si cal.
205                 }
206             }
207         }
208     }
209     return max;
210 }
211
212
213 void inv_vect_diag(real* mat, int files)
214 {
215     {
216         for (int i = 0; i < files; i++) {
217             if (mat[i] != 0.0L)
218             {
219                 mat[i] = 1.0L / mat[i];
220             }
221         }
222     }
223 }
224
225 void transpose_square_matrix(real** A, real** B, int N)
226 {
227     for (int i = 0; i < N; i++)
228         for (int j = 0; j < N; j++)
229             B[i][j] = A[j][i];
230 }
231
232 void imprimirVector(real* vec, int n)
233 {
234     {
235         for (int i = 0; i < n; i++)
236         {
237             printf("% .8Le\t\n", vec[i]);
238         }
239         printf("\n");
240     }
241 }
242
243 void imprimir_vector_en_arxiu(real* vec, int n, FILE* arxiu)
244 {
245     {
246
247         if (fopen_s(&arxiu, "error.txt", "w") == 0) {
248             printf("Arxiu obert.\n");
249             fprintf(arxiu, "theta\t error\n");
250             for (int i = 0; i < n; i++) {
251                 fprintf(arxiu, "% .6Lf\t", i * PERIOD / n);
252                 fprintf(arxiu, "% .6Le\n", vec[i]);
253             }
254             fprintf(arxiu, "\n");
255             fclose(arxiu);
256         }
257         else {
258             printf("No s'ha obert l'arxiu.\n");
259         }
260     }
```

```
261 }
262
263 void imprimirMatriu(real** A, int n, int m)
264 {
265     {
266         int i, j;
267         printf("\n");
268         for (i = 0; i < n; i++)
269         {
270             for (j = 0; j < m; j++)
271             {
272                 printf("% .6Le\t", A[i][j]);
273             }
274             printf("\n");
275         }
276         printf("\n");
277     }
278 }
279
280 void intercanviar_columnes(real** matriu, int columna_n, int columna_m, int
    N)
281 {
282     {
283         real aux;
284
285         // Intercanviar elements de les columnes
286         for (int i = 0; i < N; i++) {
287             aux = matriu[i][columna_n];
288             matriu[i][columna_n] = matriu[i][columna_m];
289             matriu[i][columna_m] = aux;
290         }
291     }
292 }
293
294 void assignar_vect_a_col(real** matriu, real* vector, int columna, int files
    )
295 {
296     {
297         for (int fila = 0; fila < files; fila++) {
298             matriu[fila][columna] = vector[fila];
299         }
300     }
301 }
302
303 void assignar_vect_a_fila(real** matriu, real* vector, int fila, int
    columnes)
304 {
305     {
306         for (int col = 0; col < columnes; col++) {
307             matriu[fila][col] = vector[col];
308         }
309     }
310 }
311
312 void reordenar_cols(real** A, int* ordre_nou, int fil, int col)
313 {
314     {
315         real** B;
316         B = crearMat(fil, col);
```

```
317         int nova_col;
318         for (int i = 0; i < col; i++) {
319             nova_col = ordre_nou[i];
320             for (int j = 0; j < fil; j++) {
321                 B[j][i] = A[j][nova_col];
322             }
323         }
324 
325         // Copiar matriu temporal a l'original
326         for (int i = 0; i < fil; i++) {
327             for (int j = 0; j < col; j++) {
328                 A[i][j] = B[i][j];
329             }
330         }
331         alliberarMat(B, fil);
332     }
333 }
334 
335 void alliberarMat(real** matriu, int files)
336 {//allibera la memoria assignada per a la matriu
337     {
338         for (int i = 0; i < files; i++)
339         {
340             free(matriu[i]);
341         }
342         free(matriu);
343     }
344 }
345 
346 void imprimir_vector_en_dues_col_arxiu(real* vec, int n, FILE* arxiu)
347 {
348     {
349 
350         if (fopen_s(&arxiu, "error_ham_aux.txt", "w") == 0) {
351             printf("Arxiu obert.\n");
352             fprintf(arxiu, "theta\t error\n");
353             for (int i = 0; i < n; i++) {
354 
355                 if(i%2==0)
356                 {
357                     fprintf(arxiu, "% .6Lf\t", i * PERIOD / n);
358                     fprintf(arxiu, "% .6Le\t", vec[i]);
359                 }else
360                 {
361                     fprintf(arxiu, "% .6Le\n", vec[i]);
362                 }
363             }
364             fprintf(arxiu, "\n");
365             fclose(arxiu);
366         }
367         else {
368             printf("No s'ha obert l'arxiu.\n");
369         }
370     }
371 }
372 
373 void imprimirMatriu_col_reals_col_im(real** A, int m, int n, FILE* arxiu)
374 {
375     int i, j;
```

```
376     if (fopen_s(&arxiu, "cursAF1.txt", "w") == 0) {
377         //printf("Arxiu obert.\n");
378         fprintf(arxiu, "theta\t real\t imaginari\n");
379         for (i = 0; i < m; i++) {
380             fprintf(arxiu, "%i\n ", i);
381             for (j = 0; j < n; j++) {
382                 fprintf(arxiu, "% .6Lf\t", j * PERIOD / n);
383                 fprintf(arxiu, "% .6Le\t", A[i][j]);
384                 fprintf(arxiu, " % .6Le\n", A[i][n+j]);
385             }
386 
387             fprintf(arxiu, "\n");
388         }
389         fprintf(arxiu, "\n");
390         fclose(arxiu);
391     }
392     else {
393         printf("No s'ha obert l'arxiu.\n");
394     }
395 }
```

Codi 5: Definició de funcions auxiliars

Acrònims

DFT Discret Fourier Transform. 31–34

EDO Equació Diferencial Ordinària. 4, 10, 27, 28

FFT Fast Fourier Transform. 34

JWST James Webb Space Telescope. 2, 3

RTBP Restricted Three Body Problem. 2, 12, 16, 43, 46

SR Sistema de Referència. 12–14

SRI Sistema de Referència Inercial. 1, 13, 17

SRR Sistema de Referència Rotatori. 13, 15, 17

SVD Singular Value Descomposition. 42, 43

TBP Three Body Problem. 12

VAP Valor Propi. 23–28

VEP Vector Propi. 23, 24, 27