CÀLCUL NUMÈRIC:

Control discret de trajectòries mitjançant maniobres impulsives

Abel Serrat Castella

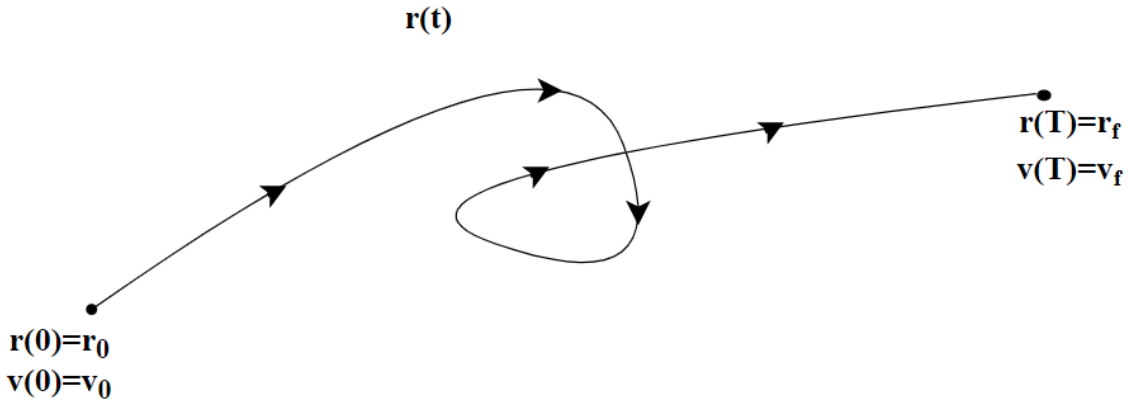
Isaac San José Couremetis

Carles Trullàs Fernàndez

Índex

1 Introducció teòrica. El problema restringit dels tres cossos (RTBP)

Comencem primer per una situació general. Suposem que tenim una partícula P a $\mathbb{R}^m$. Anomenem $r(t) = (r_1(t), \ldots, r_m(t)) \in \mathbb{R}^m$ la posició de la partícula P a l'instant t i $v(t) = (v_1(t), \ldots, v_m(t)) \in \mathbb{R}^m$ la velocitat de la partícula a temps t. Aleshores, l'objectiu és el següent: Donat un temps de vol $T > 0$, dos punts $r_0, r_f \in \mathbb{R}^m$ (posició inicial i posició final) i dues velocitats $v_0, v_f \in \mathbb{R}^m$ (velocitat inicial i velocitat final) volem trobar funcions $r(t), v(t)$ amb imatge a $\mathbb{R}^m$ i amb $t \in [0, T]$ satisfent que $r(0) = r_0$, $v(0) = v_0$, $r(T) = r_f$ i $v(T) = v_f$.



Obviament, llevat de situacions trivials, les funcions $r(t), v(t)$ hauran de complir certes restriccions que vindran donades per EDO's. Aquesta situació és justament la que nosaltres adoptarem aquí, és a dir, suposarem que el nostre moviment està "governat" per unes EDO's autònomes del tipus:

$$\begin{pmatrix} r'(t) \\ v'(t) \end{pmatrix} = f \begin{pmatrix} r(t) \\ v(t) \end{pmatrix}$$

L'espai de la variable r l'anomenarem **espai de configuracions** i l'espai total (r, v) l'anomenarem **espai de fase**, tal i com és habitual en teoria qualitativa d'equacions diferencials.

A la vida real, la situació anterior es dóna, per exemple, quan la partícula P introduïda anteriorment és un cos artificial (ja sigui una nau, un satèl·lit artificial o una sonda interplanetària). En aquests casos, nosaltres podem intentar "disenyar" les

funcions $r(t)$ i $v(t)$ de manera que es compleixin les condicions requerides. Per altra banda, és clar que no té cap sentit intentar dissenyar trajectòries de cossos cel·lestes naturals, ja que nosaltres no tenim cap mena de control sobre el seu moviment. En el cas dels cossos celestes artificials, hi ha essencialment dos mètodes per dissenyar trajectòries: la **propulsió química** i la **propulsió iònica**. Nosaltres, tal i com explicarem en detall després, ens centrarem en la propulsió química, que a grans trets es tracta de dissenyar la trajectòria mitjançant canvis en la velocitat instantanis, és a dir, canvis de velocitat en els quals no hi hagi canvi de posició. Després precisarem molt més el que estem dient.

Abans de continuar, centrem-nos en un cas particular del problema anterior: El problema restringit dels tres cossos (RTBP, Restricted Three Body Problem). Considerem $\mathbb{R}^3$ amb coordenades $(\widetilde{x}, \widetilde{y}, \widetilde{z})$. Suposem també que tenim dos cossos celestes de masses $m_1 > 0$ i $m_2 > 0$, amb $m_1 \geq m_2$. Per simplicitat, als dos cossos celestes anteriors els denotarem m_1 i m_2 respectivament i els anomenarem **cossos primaris**. Ajustant les unitats, podem suposar sense pèrdua de la generalitat que $m_1 + m_2 = 1$. Ara, suposarem a més que els cossos m_1 i m_2 orbiten circularment i de forma diametralment oposada al voltant del centre de masses (CM) entre els dos cosos, el qual sense pèrdua de la generalitat col·locarem a l'origen. Denotem $\mu = m_2$ (i per tant $\mu - 1 = -m_1 < 0$). Aleshores, en aquesta situació tenim que:

$$\widetilde{r}_1(t) = \mu(\cos(t), \sin(t), 0) \;\; i \;\; \widetilde{r}_2(t) = (\mu - 1)(\cos(t), \sin(t), 0)$$

on $\widetilde{r}_1(t)$ denota la posició del cos m_1 a l'instant $t0$ i $\widetilde{r}_2(t)$ denota la posició del cos m_2 a l'instant t. Observem que a l'instant $t = 0$, el cos m_1 es troba a la posició

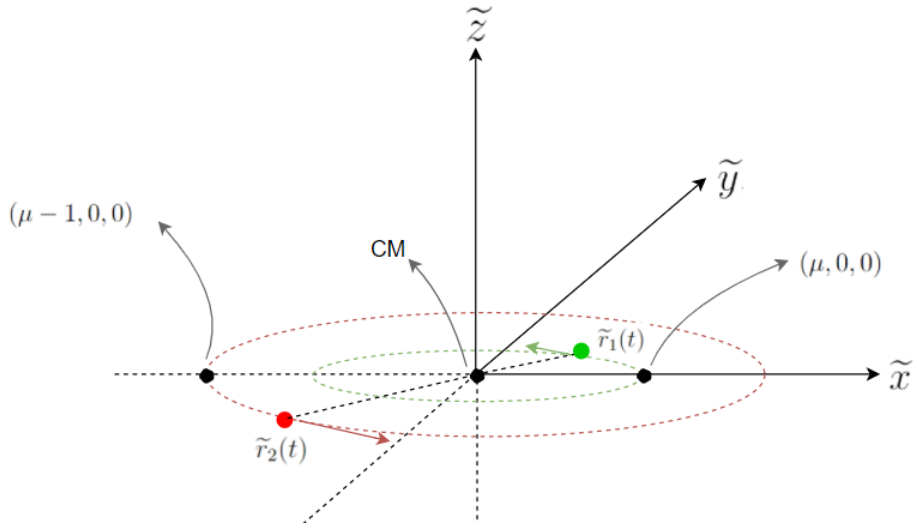
$$\widetilde{r}_1(0) = (\mu, 0, 0)$$

mentre que a $t = 0$ el cos m_2 es troba a la posició

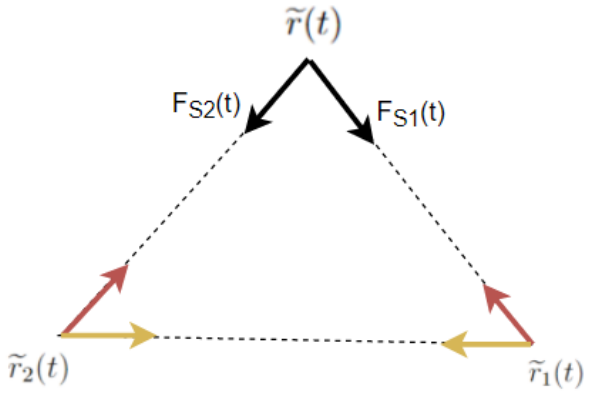
$$\widetilde{r}_2(0) = (\mu - 1, 0, 0)$$

Notem que el moviment dels cossos m_1 i m_2 està completament contingut al pla $\widetilde{x} - \widetilde{y}$. Per altra banda, tot i que ja ho hem comentat anteriorment, remarquem amb èmfasi que a cada temps t la posició dels cossos m_1 i m_2 és diametralment oposada

respecte l'origen (que és el centre de masses CM). La imatge següent descriu la situació anterior:



Observem que $|\mu - 1| \geq |\mu|$, fet que físicament té sentit ja que $m_1 \geq m_2$ i per tant el cos m_1 ha d'estar a menys distància del CM que el cos m_2. Suposem ara que tenim un satèl·lit artificial $\mathcal{F}$ de massa m (molt petita en comparació amb m_1 i m_2, essencialment negligible) sota l'acció gravitatòria de m_1 i m_2. Al satèl·lit $\mathcal{F}$ l'anomenem per simplicitat m. Ara, denotem per $\widetilde{r}(t)$ la posició del satèl·lit m a l'instant t. Anem a deduir les equacions que ha de complir $\widetilde{r}(t)$. Remarquem que tot i que el moviment dels cossos primaris estigui restringit al pla $\widetilde{x}-\widetilde{y}$, el moviment del satèl·lit m no té perquè estar-ho. Ara, la situació és la següent:



on remarquem que les forces pintades amb color vermell són negligibles degut a que el valor m és molt petit en comparació amb m_1 i m_2. Ara, com a conseqüència de

la segona llei de Newton tenim que:

$$\begin{cases} \widetilde{r}\,'(t) = \widetilde{v}(t) \\ m\widetilde{v}\,'(t) = F_{S1}(t) + F_{S2}(t) \end{cases} \tag{1}$$

A més, per la llei de la gravitació universal tenim que per $i = 1, 2$, es compleix que:

$$F_{Si}(t) = \frac{m \cdot m_i}{\|\widetilde{r}(t) - \widetilde{r}_i(t)\|^3} \left(\widetilde{r}_i(t) - \widetilde{r}(t)\right)$$

on prenent unitats adequades es pot suposar que la constant de gravitació universal G val 1. Així doncs, ens queden les equacions següents:

$$\begin{cases} \widetilde{r}\,'(t) = \widetilde{v}(t) \\ m\widetilde{v}\,'(t) = \frac{m \cdot m_1}{\|\widetilde{r}(t) - \widetilde{r}_1(t)\|^3} \left(\widetilde{r}_1(t) - \widetilde{r}(t)\right) + \frac{m \cdot m_2}{\|\widetilde{r}(t) - \widetilde{r}_2(t)\|^3} \left(\widetilde{r}_2(t) - \widetilde{r}(t)\right) \end{cases} \tag{2}$$

Finalment, observem que per a cada t fixat es compleix que:

$$\widetilde{v}\,'(t) = \frac{1-\mu}{\|\widetilde{r}(t) - \widetilde{r}_1(t)\|^3} \left(\widetilde{r}_1(t) - \widetilde{r}(t)\right) + \frac{\mu}{\|\widetilde{r}(t) - \widetilde{r}_2(t)\|^3} \left(\widetilde{r}_2(t) - \widetilde{r}(t)\right) = -\nabla \widetilde{\Omega}_t(\widetilde{r}(t))$$

on

$$\widetilde{\Omega}_t(\widetilde{x}, \widetilde{y}, \widetilde{z}) := -\left(\frac{1-\mu}{\|(\widetilde{x}, \widetilde{y}, \widetilde{z}) - \widetilde{r}_1(t)\|} + \frac{\mu}{\|(\widetilde{x}, \widetilde{y}, \widetilde{z}) - \widetilde{r}_2(t)\|} \right)$$

Aquest últim fet sera necessari a posteriori.

Canvi a coordenades sinòdiques

Un cop trobades les equacions del satèl·lit artificial m sota l'acció gravitatòria de m_1 i m_2, ens agradaria simplificar-les mitjançant algun canvi de variables intel·ligent. Una manera de procedir és la següent: Si recordem les expressions de $\widetilde{r}_1(t)$ i $\widetilde{r}_2(t)$ teníem que $\widetilde{r}_1(t) = \mu(\cos(t), \sin(t), 0)$ i que $\widetilde{r}_2(t) = (\mu - 1)(\cos(t), \sin(t), 0)$. Ara, considerem la següent rotació[1]:

$$R(t) := \begin{pmatrix} \cos(t) & -\sin(t) & 0 \\ \sin(t) & \cos(t) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

[1]De fet, parlant amb precisió, tenim que per a cada t fixat, la matriu $R(t)$ és una rotació d'angle t. No obstant això, acceptarem aquest abús de llenguatge.

Aleshores, aplicant la rotació $R(t)^{-1}$ a les trajectòries $\widetilde{r_1}(t)$ i $\widetilde{r_2}(t)$ obtenim que:

$$\mu \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = R(t)^{-1}\widetilde{r_1}(t)^T \quad i \quad (\mu - 1) \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = R(t)^{-1}\widetilde{r_2}(t)^T$$

És a dir, aplicant la rotació $R(t)^{-1}$, les trajectòries dels cossos primaris queden fixades a la posició inicial. Això és una simplificació important i útil. Seguint amb aquesta idea, ara el que farem és considerar el canvi de variables $r(t) = R(t)^{-1}\widetilde{r}(t)$[2], on $r(t)$ és la posició del cos m a temps t en les noves coordenades. Les noves coordenades les denotarem simplement per (x, y, z) i les anomenarem *coordenades sinòdiques*. Per deduir les equacions que ha de complir $r(t)$ només ens cal aplicar el canvi de variables $r(t) = R(t)^{-1}\widetilde{r}(t)$ a (2). Abans de fer-ho, observem que el sistema (2) no és autònom, però el sistema que obtindrem amb el canvi de variables sí que ho serà! Aquest fet és molt important i ens simplificarà molt l'estudi. Passem a fer els càlculs explícits per tal d'expressar el sistema (2) en les noves coordenades (coordenades sinòdiques). Denotant $A(t) := R(t)^{-1}$ tenim que:

$$\begin{cases} r'(t) = A'(t)\widetilde{r}(t) + A(t)\widetilde{r}\,'(t) := v(t) \\ v'(t) = A''(t)\widetilde{r}(t) + 2A'(t)\widetilde{r}\,'(t) + A(t)\widetilde{r}\,''(t) \end{cases} \tag{3}$$

Ara, ens cal escriure l'expressió de $v'(t)$ a (3) en termes de $v(t)$ i de $r(t)$. Per fer-ho, observem que:

$$\widetilde{r}\,'(t) = A(t)^{-1}(v(t) - A'(t)R(t)r(t))$$

Per altra banda, observem que:

$$\widetilde{r}\,''(t) = \widetilde{v}\,'(t) = -\nabla\widetilde{\Omega}_t(\widetilde{r}(t)) = -\nabla\widetilde{\Omega}_t(R(t)r(t))$$

A més, tenim que:

$$\|R(t)r(t) - \widetilde{r_1}(t)\| = \|R(t)\|\,\|r(t) - (\mu, 0, 0)\| = \|r(t) - (\mu, 0, 0)\|$$

$$\|R(t)r(t) - \widetilde{r_2}(t)\| = \|R(t)\|\,\|r(t) - (\mu - 1, 0, 0)\| = \|r(t) - (\mu - 1, 0, 0)\|$$

[2]Estrictament parlant, hauríem de posar $r(t)^T = R(t)^{-1}\widetilde{r}(t)^T$. Tot i així, a partir d'ara quan no hi hagi perill de confusió, ometrem els símbols de transposició T.

on hem usat que $\|R(t)\| = 1$ ja que $R(t)$ és una rotació per a cada t. A partir de les dues últimes igualtats és directe veure que:

$$\widetilde{\Omega}_t(\widetilde{r}(t)) = \widetilde{\Omega}_t(R(t)r(t)) = -\frac{1-\mu}{\|r(t)-(\mu,0,0)\|} - \frac{\mu}{\|r(t)-(\mu-1,0,0)\|}$$

Finalment, a partir de totes les relacions que hem anat obtenint s'arriba fàcilment a les equacions que estan escrites a l'enunciat de la pràctica, és a dir:

$$\begin{cases} r' = v \\ v' = a(r,v) \end{cases} \tag{4}$$

on, si denotem $r = (x,y,z)$ i $v = (u,v,w)$,

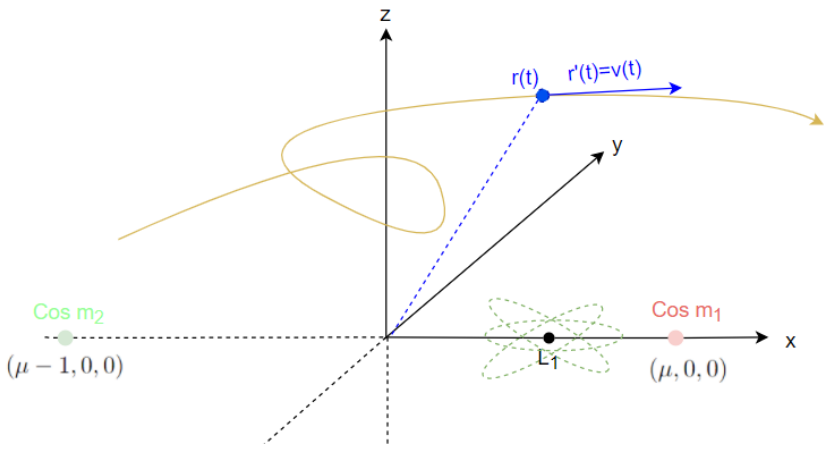
$$a(r,v) = 2(v,-u,0) + \nabla\Omega(r)$$

$$\Omega(r) = \frac{x^2+y^2}{2} + \frac{1-\mu}{\rho_1} + \frac{\mu}{\rho_2} - \frac{1}{2}\mu(1-\mu)$$

$$\rho_1(r) = \sqrt{(x-\mu)^2+y^2+z^2}$$

$$\rho_2(r) = \sqrt{(x-\mu+1)^2+y^2+z^2}$$

Per tant, observem que en particular el sistema (4) és autònom. Remarquem també que les unitats de temps són tals que els cossos primaris completen una revolució en 2π unitats de temps. Per exemple, en el cas terra-lluna 2π unitats de temps corresponen a un mes. En el cas terra-sol corresponen a un any. En les noves coordenades, la situació és la següent:



Observem que ara la novetat és que els cossos primaris m_1 i m_2 estan fixats als eixos. Per altra banda, el satèl·lit descriu una trajectòria a l'espai tridimensional $\mathbb{R}^3$.

Abans de seguir, anem a fer una observació interessant: El punt L_1 que hem dibuixat a l'anterior figura és un punt fix del sistema (4) i s'anomena **punt fix Lagrangià**. Si suposem per exemple que m_2 és la terra i que m_1 és el sol, el punt L_1 seria un lloc perfecte per enviar-hi un observatori solar. Bueno... perfecte del tot no, ja que l'observatori no pot estar completament estàtic a L_1 degut a que impediria la comunicació per radio amb la terra. Per sort, es pot demostrar que en un entorn del punt crític L_1 existeixen òrbites periòdiques que envolten L_1 (al dibuix estan pintades en verd) anomenades **òrbites Halo**. Aquestes trajectòries sí que són ideals per col·locar-hi un observatori solar. De fet, en l'actualitat la missió SOHO es troba en una d'aquestes òrbites.

Reprenem ara el problema (general) que hem plantejat inicialment. El restringirem lleugerament. Concretament, considerarem la següent situació: Donades posicions i velocitats inicials $(r_0, v_0) \in \mathbb{R}^m \times \mathbb{R}^m$, posicions i velocitats finals $(r_f, v_f) \in \mathbb{R}^m \times \mathbb{R}^m$ i un temps de vol Δt, volem trobar maniobres (canvis instantanis en la velocitat) Δv_0 i Δv_1 tals que, partint de (r_0, v_0), en aplicar Δv_0, fer vol lliure durant $\Delta t/2$ unitats de temps, aplicar Δv_1, i fer vol lliure durant $\Delta t/2$ unitats de temps més, acabem amb posició i velocitat (r_f, v_f). Recordem que estàvem suposant també que el moviment estava governat per una EDO autònoma de primer ordre de la forma:

$$\begin{pmatrix} r'(t) \\ v'(t) \end{pmatrix} = f \begin{pmatrix} r(t) \\ v(t) \end{pmatrix}$$

Aleshores, si denotem com $\phi_t(r, v) := \phi(t, 0, (r, v))$ el flux de l'EDO anterior, aleshores el procés que d'anar de (r_0, v_0) a (r_f, v_f) tot aplicant les maniobres Δv_0, Δv_1 el podem formular així:

$$(r_0, v_0) \overset{mani}{\to}^{\Delta v_0} (r_0, v_0 + \Delta v_0) \overset{vol\ \ lliure}{\to} \phi_{\Delta t/2}(r_0, v_0 + \Delta v_0)$$
$$\overset{mani}{\to}^{\Delta v_1} \phi_{\Delta t/2}(r_0, v_0 + \Delta v_0) + (0, \Delta v_1) \overset{vol\ \ lliure}{\to} \phi_{\Delta t/2}(\phi_{\Delta t/2}(r_0, v_0 + \Delta v_0) + (0, \Delta v_1))$$

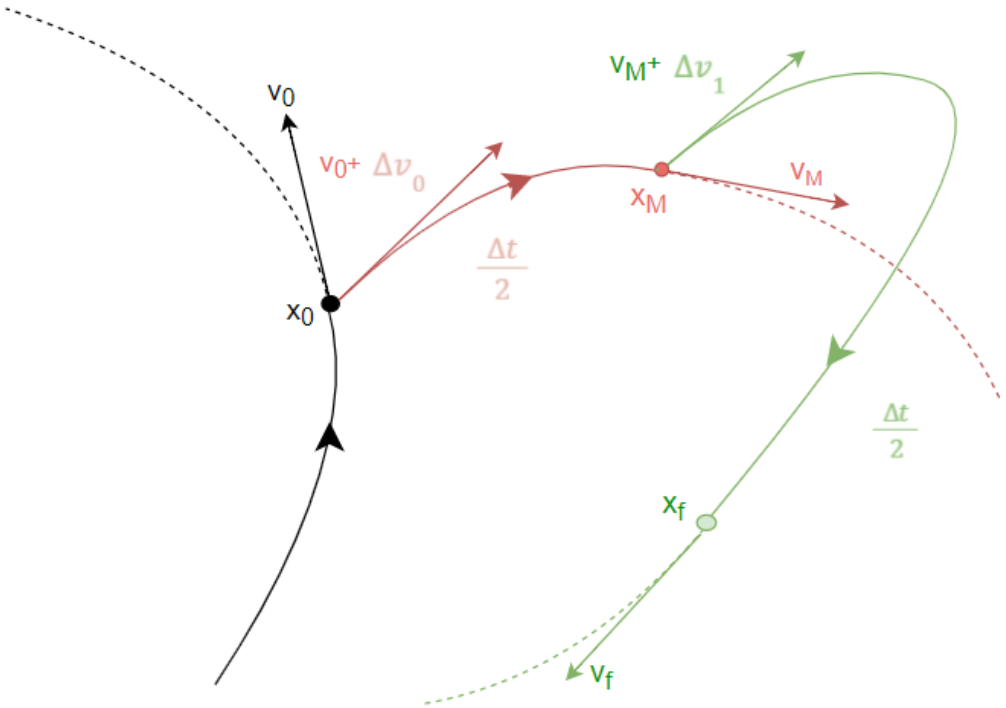
Com que volem que la darrera expressió sigui igual a (r_f, v_f), volem trobar maniobres

$$\Delta v = (\Delta v_0, \Delta v_1) \in \mathbb{R}^n = \mathbb{R}^m \times \mathbb{R}^m$$

que anul·lin la funció $G : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ definida com:

$$G(\Delta v) = \underbrace{\phi_{\Delta t/2}(\underbrace{\phi_{\Delta t/2}\underbrace{(r_0, v_0 + \Delta v_0)}_{:=x_{0.5}} + (0, \Delta v_1)}_{:=x_{1.5}}) - (r_f, v_f)} \tag{5}$$

Gràficament, la situació és la següent:



Ara, observem que l'expressió $G(\Delta v) = 0$ és una equació no lineal de $n = 2m$ equacions. Una manera de resoldre-la és usant el **Mètode de Newton**. Per implementar-lo, la major dificultat que ens sorgirà serà el càlcul de la diferencial de G. Per fer-ho, observem que:

$$DG(\Delta v) = (D_{\Delta v_0} G(\Delta v) \mid D_{\Delta v_1} G(\Delta v)) \tag{6}$$

A més, tenim que:

$$D_{\Delta v_0} G(\Delta v) = D\phi_{\Delta t/2}(x_{1.5}) D_v \phi_{\Delta t/2}(x_{0.5}) \;\; i \;\; D_{\Delta_{v_1}} G(\Delta v) = D_v \phi_{\Delta t/2}(x_{1.5})$$

Tenint presents aquestes relacions, veurem que les eines que desenvoluparem durant la pràctica ens permetran implementar de forma factible el mètode de Newton al nostre problema. Finalment, observem que en particular el procés que acabem de descriure també és vàlid per al RTBP.

Tal i com veurem a continuació, tots els apartats d'aquesta pràctica seran petits passos a seguir per tal d'aconseguir resoldre el problema final que acabem de descriure. Essencialment, el procés serà: A l'apartat 1 construirem un integrador d'EDO's, a l'apartat 2 programarem l'implementació de les equacions variacionals, a l'apartat 3 resoldrem sistemes lineals sobre-determinats i finalment a l'apartat 4 recollirem totes les eines que hem anat creant per poder resoldre el problema que hem comentat abans (mitjançant el mètode de Newton), el qual engloba com a cas particular el RTBP.

2 Resolució del RTBP per passos

A continuació, exposem la resolució de la pràctica, completant els exercicis continguts en cadascun dels 4 apartats que disposa el guió.

Abans però, expliquem l'estructura dels fitxers i codis per fer possible un seguiment coordinat de la resolució dels exercicis amb les eines emprades per resoldre'ls. L'estructura dels fitxers ve donada per l'esquema 1. De manera resumida, expliquem què conté cada carpeta.

La carpeta principal, tal com podem veure del diagrama 1, és `python_version` i conté dues carpetes principals, `scripts` i `outputs`. Dins de `scripts`, hi ha continguts els codis que resolen els exercicis dels 4 apartats de la pràctica. Els primers dos caràcters dels noms dels fitxers indiquen l'apartat que resolen i la resta, l'exercici. Aquests fitxers fan ús d'utilitats que es requereixen al llarg de la pràctica, i per tant, tenim la carpeta `utils` que conté fitxers amb utilitats. Si l'objectiu d'un exercici concret és construir una utilitat, tot i ser un fitxer principal, el posicionarem dins d'aquesta carpeta. A més, els fitxers principals sovint requereixen llegir i crear fitxers que contenen dades i són específics de l'exercici. Per tant, tenim la carpeta `outputs`, que de manera ordenada conté els fitxers d'entrada i de sortida i les eines

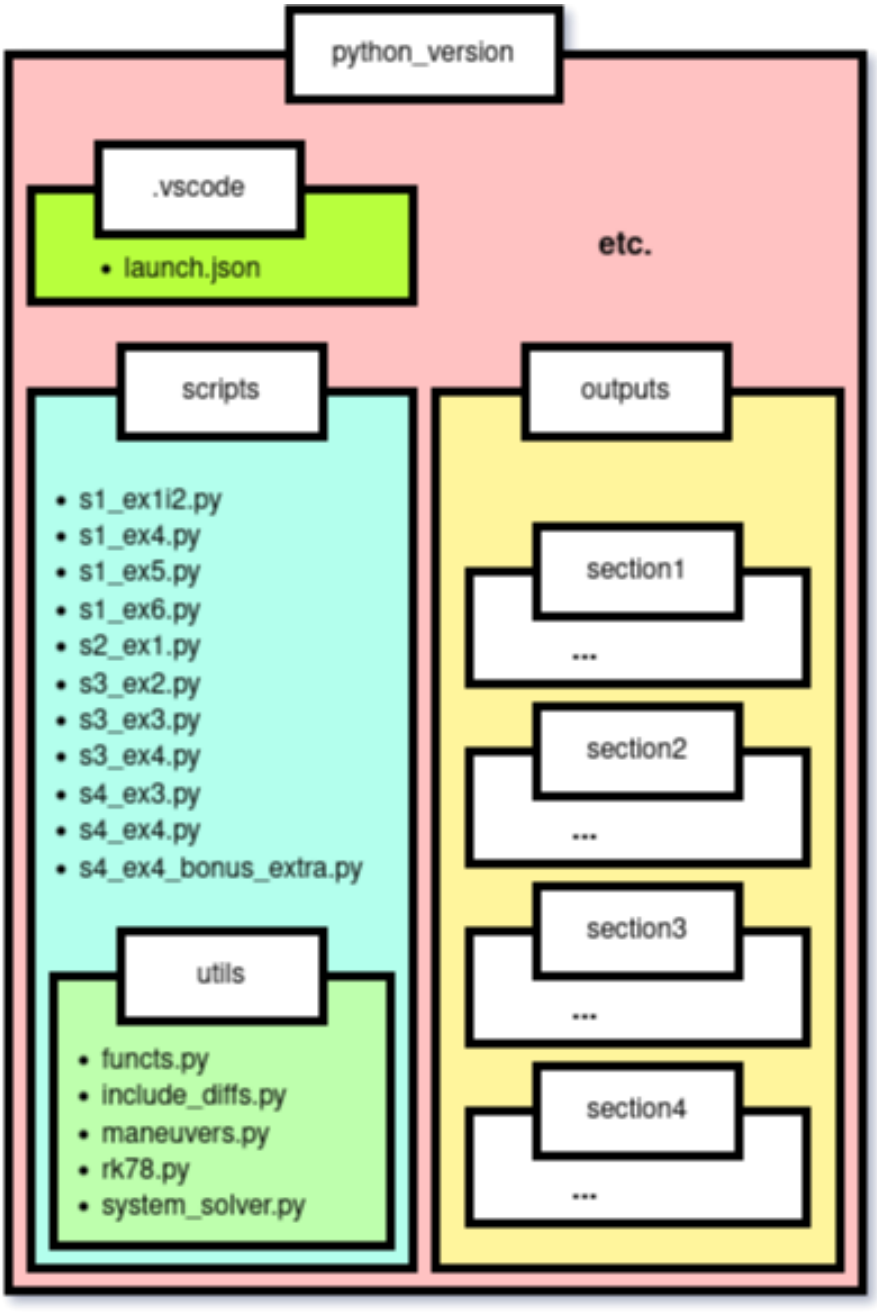


Figura 1: Estructura dels fitxers.

necessàries si s'escau per generar gràfics. Últim però no menys important, per a l'execució del codi hem utilitzat l'eina **Visual Studio Code**, que genera el fitxer `.vscode/launch.json` amb les configuracions de les execucions, per a cada programa. Identificant la configuració amb el programa que executa aquesta configuració, l'usuari pot trobar els fitxers d'entrada i sortida.

2.1 Retrat-fase i flux d'un sistema d'EDO

En aquesta secció presentarem els resultats dels exercicis sobre el retrat-fase i el flux d'un sistema d'EDO i descriurem l'estructura del codi que ens ha permès resoldre'ls i on localitzar-lo.

Exercici 1

Aquest exercici consisteix a desenvolupar un programa que implementi un mètode de Runge-Kutta-Fehlberg (RKF) d'ordres 7 i 8 a les equacions del pèndol. El programa rebrà com a arguments `h_min`, `h_max` i `tol`, llegirà un fitxer on cada línia contindrà un input de la forma

```
x0 y0 h0 np
```

i per a cada input aplicarà el mètode RKF amb `np` passos partint del punt $(\texttt{x0}, \texttt{y0})$, temps `t=0` i pas inicial `h0`. El programa haurà de mostrar, per a cada input, `np` línies (corresponents als `np` passos) de la forma

```
t x y h
```

on `t` és el temps actual, $(\texttt{x}, \texttt{y})$ la posició actual i `h` és el pas proposat. A més, caldrà deixar una línia en blanc cada cop que es passa a analitzar el següent input.

Hem creat un fitxer `scripts/s1_ex1i2.py` que duu a terme aquest procediment. Per començar, aquest fitxer rep com a arguments el path del fitxer on es troben les condicions inicials (`outputs/section1/ex1i2/condicions_inicials/condicions_basiques.txt`) i el path de la carpeta on es trobarà el fitxer que emmagatzemarà els resultats (`outputs/section1/ex1i2/orbites_output`). Aquests paths estan definits al fitxer `.vscode/launch.json`.

A continuació, el programa carrega les condicions inicials, declara i inicialitza els valors dels paràmetres `h_min`, `h_max` i `tol` (si es volen modificar, cal fer-ho aquí) i prepara la carpeta `orbites_output` on es guardaran els resultats en un fitxer `orbites.txt`. En cas que la carpeta no existeixi es crea des de zero, i en cas que el fitxer ja existeixi s'esborra el seu contingut.

Finalment, per a cada condició inicial (per a cada input del fitxer `condicions_inicials.txt`), el programa crida a la funció `rk78_multistep()` del fitxer `scripts/utils/rk78.py` (mitjançant una instància de la classe `rk78`) passant tots els paràmetres de l'input i passant la funció que implementa el camp vectorial de les equacions del pèndol (la qual està definida al fitxer `scripts/utils/functs.py`) i emmagatzema els outputs obtinguts seguint el format que hem descrit abans. Internament, la funció `rk78_multistep()` s'encarrega de cridar `np` vegades a la funció `rk78()` (del mateix fitxer), la qual implementa un pas del mètode de RKF i retorna el temps final `t` del pas, la posició final $(\texttt{x}, \texttt{y})$ del pas i el nou pas proposat `h`. Amb aquesta informació, la funció `rk78_multistep()` retorna un array amb els outputs de les `np` crides a la funció `rk78()`.

Exercici 2

L'execució del programa `s1_ex1i2.py` retorna un fitxer `orbites.txt` amb els resultats d'integrar les equacions del pèndol amb diferents condicions incials. Ara ens interessa representar un retrat-fase del pèndol a partir de les òrbites obtingudes al fitxer `orbites.txt`. Hem creat un fitxer `outputs/section1/ex1i2/comandes_gnuplot.txt` amb les instruccions que hem utilitzat per generar el retrat-fase amb gnuplot. La Figura 2 mostra la gràfica obtinguda fixant els següents valors per als paràmetres:

- $(\texttt{h\_min}, \texttt{h\_max}) = (0.01, 0.05)$ (pas mínim i pas màxim permesos)
- $\texttt{tol} = 0.0001$ (tolerància)

Observem que algunes òrbites semblen incompletes. Això es deu a que totes les òrbites s'han obtingut partint del temps `t=0` i fent `np` passos del mètode RKF, de manera que el mètode no contempla valors negatius o valors positius molt grans de `t`. Per tal de corregir-ho, hem creat un nou fitxer de condicions inicials `outputs/section1/ex1`

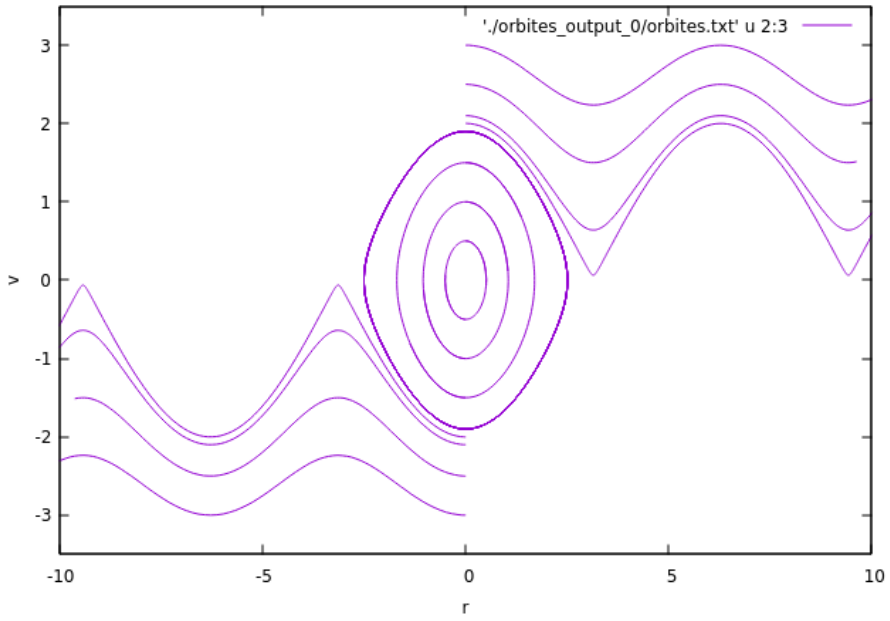


Figura 2: Retrat-fase del pèndol (incomplet)

`i2/condicions_inicials/condicions_totals.txt` on hem afegit nous inputs als ja existents per completar les òrbites que es veuen incompletes a la gràfica. D'aquesta manera, si canviem el path del fitxer de lectura en el fitxer `.vscode/launch.json` i executem el programa `s1_ex1i2.py`, podem obtenir un nou fitxer `orbites.txt` amb les òrbites completes. La Figura 3 mostra la gràfica obtinguda amb gnuplot a partir dels nous resultats.

Exercici 3

L'inconvenient principal de la funció `rk78_multistep()` és que aquesta funció implementa un nombre fixat de passos del mètode RKF i, per tant, no tenim cap control sobre l'últim valor de `t` en el qual s'integrarà l'EDO, ja que la longitud de cada pas va canviant segons uns criteris fixats pel mètode RKF. Per aquesta raó, aquesta funció és insuficient si volem estimar el valor d'una solució d'una EDO en un instant `t` determinat.

Per tant, cal desenvolupar una nova funció que implementi tants passos del mètode RKF com sigui necessari per assolir l'instant `t` en què volem estimar la solució, de manera que la funció ens informi sobre l'èxit o el fracàs de l'estimació segons si

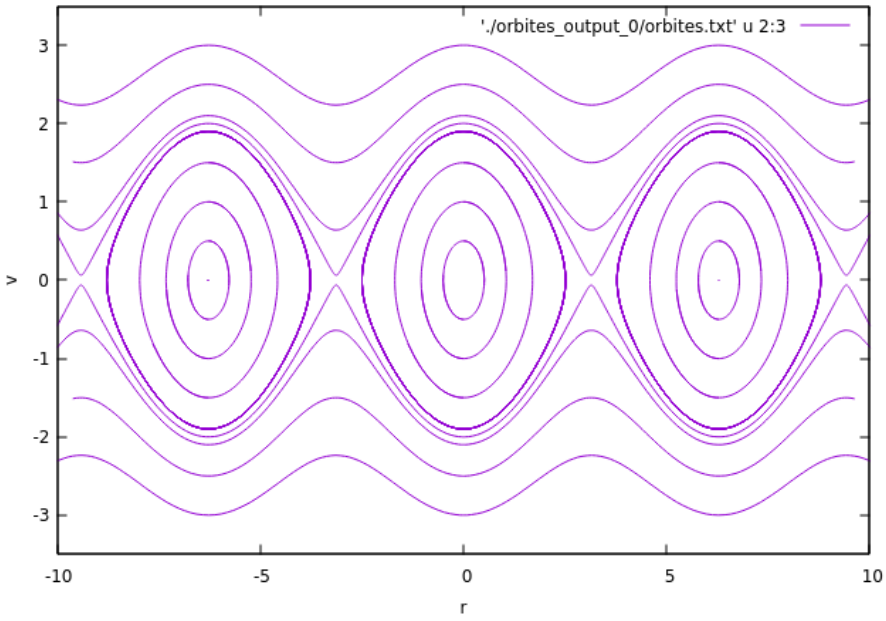


Figura 3: Retrat-fase del pèndol (complet)

s'ha aconseguit assolir l'instant `t` en menys passos que un cert límit fixat o no.

Hem creat una funció `flux()` en el fitxer `scripts/utils/rk78.py` per tal d'aconseguir aquest objectiu. Aquesta funció rep com a arguments les condicions inicials i els paràmetres que s'utilitzaran en el mètode de RKF, la funció que implementa el camp vectorial de l'EDO que es vol integral, l'instant `t_pred` en què es vol estimar la solució i el nombre màxim de passos permesos. Per començar, la funció comprova la posició relativa de `t0` (instant inicial) i `t_pred` i:

- Si `t_pred` = `t0`, s'atura la funció (retorn immediat), ja que el valor de la solució que es vol estimar és el valor inicial `x0`.
- Si `t_pred` > `t0`, s'agafa com a pas inicial del mètode de RKF el valor que rep la funció com a argument.
- Si `t_pred` < `t0`, s'agafa com a pas inicial del mètode de RKF el valor que rep la funció com a argument però canviat de signe (caldrà implementar el mètode cap enrere en el temps).

A continuació, es duen a terme crides successives a la funció `rk78()` (la qual implementa un pas del mètode de RKF a cada crida) i es van emmagatzemant els outputs

obtinguts fins que s'assoleix o es supera l'instant de temps `t_pred` o bé fins que el nombre de crides assoleix el nombre màxim de passos permesos.

Finalment,

- Si s'ha assolit el nombre màxim de passos però no s'ha assolit l'instant `t_pred`, aleshores la funció retorna un missatge de fracàs (en forma de variable binària/booleana).
- Si s'ha assolit o s'ha superat l'instant `t_pred`, aleshores la funció esborra els resultats de l'última crida a `rk78()` i torna a cridar a la funció `rk78()` un cop més agafant com a longitud de pas la diferència entre `t_pred` i l'instant de temps `t` obtingut en la penúltima crida que s'ha fet. D'aquesta manera, s'aconsegueix l'estimació del valor de la solució en l'instant `t_pred` i la funció retorna un missatge d'èxit (en forma de variable binària/booleana) juntament amb el valor estimat de la solució.

Exercici 4

A continuació, verificarem la funció `flux()` que hem creat integrant el problema de valors inicials

$$\begin{cases} y' = 2y/t \\ y(1) = 1 \end{cases}$$

que té per solució $y(t) = t^2$, i integrant també l'oscil·lador harmònic

$$\begin{cases} x' = y \\ y' = -x \end{cases}$$

amb condicions inicials $(x(0), y(0)) = (1, 0)$, que té per solució $(x(t), y(t)) = (cos(t), -sin(t))$.

Per tal de verificar-ho, establirem com a temps d'estimació `t_pred` $= 3.2506$ i fixarem els següents valors per als paràmetres:

- (`h_min`, `h_max`) $= (0.01, 0.05)$ (pas mínim i pas màxim permesos)
- `tol` $= 0.0001$ (tolerància)
- `max_steps` $= 1000$ (nombre màxim de passos permesos)

- `h` $= 0.01$ (pas inicial proposat)

Aleshores, cal aplicar la funció `flux()` als dos problemes de valors inicials, comparar els resultats obtinguts amb les solucions reals i verificar que els errors obtinguts són coherents amb la tolerància que hem fixat.

Hem creat un fitxer `scripts/s1_ex4.py` on es duu a terme aquesta comprovació. Per començar, es declaren i s'inicialitzen els valors dels paràmetres, de les condicions inicials i de `t_pred`. A continuació, per a cada PVI es crida a la funció `flux()` (mitjançant una instància de la classe `rk78`) passant les condicions inicials i tots els paràmetres i passant també la funció que implementa el camp vectorial del PVI corresponent (la qual està definida al fitxer `scripts/utils/functs.py`). Finalment, per a cada PVI s'imprimeix l'output retornat per `flux()` i, en cas que aquesta funció hagi retornat un missatge d'èxit en l'estimació, s'imprimeix un missatge informant de si les diferències entre les components de les solucions obtingudes (estimacions en l'instant `t_pred`) i les components de les solucions reals (obtingudes en avaluar les solucions reals/exactes en $t =$ `t_pred`) són menors (en valor absolut) que la tolerància `tol` o no.

Mostrem a continuació els resultats obtinguts en executar aquest programa:

```
{'success': True, 'pos': array([10.56640036])}
Does the estimation satisify the tolerance?: [ True]
{'success': True, 'pos': array([-0.99406458,  0.10879159])}
Does the estimation satisify the tolerance?: [ True  True]
```

Veiem, doncs, que per a ambdós PVI la funció `flux()` ha pogut trobar una estimació del valor de la solució en l'instant `t_pred` i l'error d'aquesta estimació és menor que la tolerància fixada. Per tant, queda comprovat que la funció `flux()` funciona correctament.

Exercici 5

L'objectiu d'aquest exercici és integrar numèricament les equacions de Lorenz

$$\begin{cases} \dot{x} = \sigma(y - x) \\ \dot{y} = -xz + \rho x - y \\ \dot{z} = xy - \beta z \end{cases}$$

i representar gràficament l'atractor de Lorenz. Per tant, donades unes condicions inicials, caldrà trobar el valor de la solució corresponent en una llista ordenada d'instants de temps $t_0, t_1, \ldots, t_n$.

Una possible manera de fer això, seria cridar $n + 1$ cops a la funció `flux()` per estimar el valor de la solució en cada t_i, partint sempre de l'instant inicial t_0. Malgrat això, si volem optimitzar computacionalment aquest procés, no és bona idea començar a integrar sempre des de t_0, ja que aleshores hi hauria passos del mètode que es repetirien innecessàriament per a cada crida de la funció `flux()`. Per tant, l'estratègia que seguirem consistirà a estimar la solució en t_1 partint de t_0, després estimar la solució en t_2 partint de t_1 (utilitzant com a condició inicial l'estimació que haguem obtingut de la solució avaluada en t_1) i així successivament.

Hem creat una funció `flux_multistep()` en el fitxer `scripts/utils/rk78.py` que duu a terme aquest procés. Aquesta funció rep com a arguments les condicions inicials i els paràmetres que s'utilitzaran en el mètode de RKF, la funció que implementa el camp vectorial de l'EDO que es vol integral, l'últim instant `t_pred_lim` en què es vol estimar la solució, el nombre d'avaluacions de la solució que es volen fer (nombre de crides a la funció `flux()`) i el nombre màxim de passos permesos en cada crida a la funció `flux()`. Per començar, la funció corregeix el signe del pas inicial del mètode de RKF segons la posició relativa de `t0` (instant inicial) i `t_pred_lim`, tal com feia la funció `flux()`.

A continuació, es crea una llista d'instants de temps ordenats equidistants compresos entre `t0` i `t_pred_lim` (tants com 1 + el nombre d'avaluacions de la solució que es volen fer). Llavors, per a cada instant (sense comptar-ne l'últim) es crida a la funció `flux()` per estimar el valor de la solució en el següent instant, passant com a condicions inicials l'instant "actual" i l'estimació de la solució en aquest instant (tal com hem descrit abans).

Cada crida a `flux()` retorna un missatge d'èxit o fracàs i, en cas d'èxit, es retorna una estimació de la solució. Si alguna crida retorna un missatge de fracàs, la funció `flux_multistep()` atura aquest procés i retorna un missatge de fracàs. En cas contrari, en acabar totes les crides, la funció retorna un missatge d'èxit junta-

ment amb els valors estimats de la solució en els diferents instants de temps.

Per tal d'implementar la funció `flux_multistep()` a les equacions de Lorenz hem creat un fitxer `scripts/s1_ex5.py` amb una estructura molt similar al fitxer `scripts/s1_ex1i2.py`. Per començar, aquest fitxer rep com a argument el path de la carpeta on es trobarà el fitxer que emmagatzemarà els resultats (`outputs/section1/ex5/orbites_output`). Aquest path està definit al fitxer `.vscode/launch.json`.

A continuació, el programa declara i inicialitza els valors dels paràmetres i les condicions inicials (si es volen modificar, cal fer-ho aquí) i prepara la carpeta `orbites_output` on es guardaran els resultats en un fitxer `orbites.txt`. En cas que la carpeta no existeixi es crea des de zero, i en cas que el fitxer ja existeixi s'esborra el seu contingut.

Finalment, el programa crida a la funció `flux_multistep()` (mitjançant una instància de la classe `rk78`) passant les condicions inicials i tots els paràmetres i passant la funció que implementa el camp vectorial de les equacions de Lorenz (la qual està definida al fitxer `scripts/utils/functs.py`) i emmagatzema els outputs obtinguts (una llista de línies de text on cada línia conté 4 valors: el primer és un instant de temps i els altres tres són les components de l'estimació de la solució en aquell instant).

Per tal de representar gràficament l'atractor de Lorenz, hem executat el fitxer `s1_ex5.py` agafant-nos els següents valors dels paràmetres i condicions inicials:

- Paràmetres de l'EDO: $\sigma = 3$, $\rho = 26.5$, $\beta = 1.0$
- Paràmetres de la funció `flux_multistep()`:
  - `h` $= 0.01$ (pas inicial proposat)
  - (`h_min`, `h_max`) $= (0.01, 0.05)$ (pas mínim i pas màxim permesos)
  - `tol` $= 0.0001$ (tolerància)
  - `max_steps` $= 1000$ (nombre màxim de passos permesos en cada crida a `flux()`)
  - `num_evals` $= 10000$ (nombre d'avaluacions de la solució)

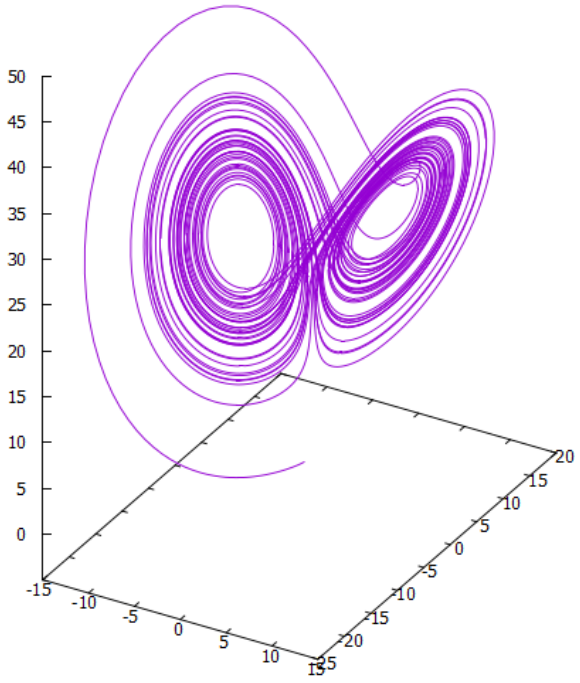


Figura 4: Atractor de Lorenz

  - `t_pred` = 100 (últim instant d'estimació de la solució)

- Condicions inicials: `t0` = 0, (`x0`, `y0`, `z0`) = (−0.416, −0.909, 0.014)

Hem creat un fitxer `outputs/section1/ex5/comandes_gnuplot.txt` amb les instruccions que hem utilitzat per representar gràficament l'atractor de Lorenz amb gnuplot. La Figura 4 mostra la gràfica obtinguda.

Exercici 6

En aquest exercici volem desenvolupar un programa similar al programa que hem aplicat a les equacions del pèndol però aplicant-lo al RTBP, per tal de representar gràficament una família d'òrbites halo al voltant de L_1 per al cas Terra-Lluna.

Per a fer-ho, hem creat un fitxer `scripts/s1_ex6.py` que té la mateixa estructura que el fitxer `s1_ex1i2.py`: per començar, rep com a arguments el path del fitxer on es troben les condicions inicials (`outputs/section1/ex6/condicions_inicials/halos.inp`) i el path de la carpeta on es guardaran els resultats (`outputs/section1/ex6/`

`orbites_output`), els quals estan definits al fitxer `.vscode/launch.json`; a continuació, el programa carrega les condicions inicials, declara i inicialitza els valors dels paràmetres i prepara la carpeta `orbites_output` i els fitxers que contindrà (esborrant el contingut existent); i finalment, per a cada condició inicial, el programa crida a la funció `rk78_multistep()` passant tots els paràmetres i passant la funció que implementa el camp vectorial de les equacions del RTBP en coordenades sinòdiques (la qual està definida al fitxer `scripts/utils/functs.py`) i emmagatzema els outputs obtinguts.

Comentem, però, els canvis (diferències) que hi ha al programa `s1_ex6.py` respecte el programa `s1_ex1i2.py`:

- Hem canviat els valors dels paràmetres pels següents:
  - $(\texttt{h\_min}, \texttt{h\_max}) = (0.0001, 0.005)$ (pas mínim i pas màxim permesos)
  - $\texttt{tol} = 0.000001$ (tolerància)

  A més, cal afegir un paràmetre `mu` corresponent al paràmetre μ de les equacions del RTBP, al qual li assignem el valor $\texttt{mu} = 0.01215058560962404$.

- Mentre que les equacions del pèndol en dimensió 2 (una dimensió per la posició i una dimensió per la velocitat) formen un sistema d'EDO 2×2, les equacions del RTBP formen un sistema d'EDO 6×6 (hi ha 6 variables: les 3 coordenades de la posició i les 3 coordenades de la velocitat). Per tant, els inputs (línies) del fitxer que conté les condicions inicials del RTBP són de la forma

  ```
  x0 y0 z0 u0 v0 w0 h0 np
  ```

- Mentre que el programa `s1_ex1i2.py` emmagatzemava en un fitxer `orbites.txt` outputs (línies) que contenien el temps actual, la posició actual i el pas proposat, el programa `s1_ex6.py` només emmagatzema les posicions (i velocitats) actuals, és a dir, emmagatzema en un fitxer `outputs/section1/ex6/orbites_output/orbites.txt` outputs de la forma

  ```
  x y z u v w
  ```

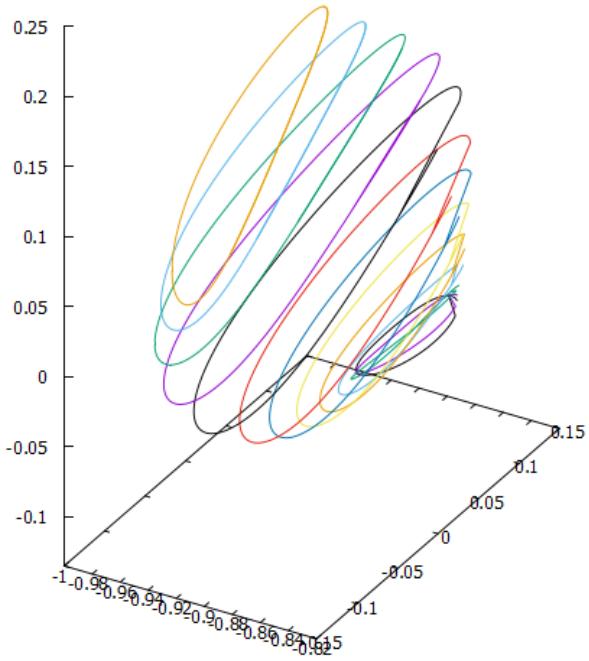


Figura 5: Òrbites halo en el sistema Terra-Lluna

A més, el programa `s1_ex6.py` també emmagatzema en un fitxer `outputs/section1/ex6/orbites_output/points.txt` les coordenades de les posicions de la Terra i la Lluna (en coordenades sinòdiques), que només depenen del paràmetre `mu`.

Hem creat un fitxer `outputs/section1/ex6/comandes_gnuplot.txt` amb les instruccions que hem utilitzat per representar la família d'òrbites halo obtingudes amb gnuplot. La Figura 5 mostra la gràfica obtinguda.

2.2 Les variacionals primeres i la diferencial del flux

En aquesta secció presentarem els resultats dels exercicis sobre les variacionals primeres i la diferencial del flux i descriurem l'estructura del codi que ens ha permès resoldre'ls i on localitzar-lo.

Exercici 1

Considerem el sistema d'EDO

$$\begin{cases} \dot{x} = \alpha(1-r^2)x - y \\ \dot{y} = x + \alpha(1-r^2)y \end{cases} \tag{7}$$

on $r^2 = x^2 + y^2$ i α és un paràmetre.

L'objectiu d'aquest exercici és desenvolupar un programa que integri numèricament aquest sistema (fixarem condicions inicials $(x(0), y(0)) = (1, 0)$) juntament amb les seves variacionals primeres.

Per tant, hem d'integrar el següent problema de valors inicials:

$$\begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}) \\ \dot{\boldsymbol{A}} = D_{\boldsymbol{x}}\boldsymbol{f}(t, \boldsymbol{x})A \\ \boldsymbol{x}(t_0) = \boldsymbol{x}_0\,,\ A(t_0) = I_2 \end{cases} \tag{8}$$

on:

- $\boldsymbol{x} = (x, y)^t \in \mathbb{R}^2$
- $\boldsymbol{f}\colon \mathbb{R}\times\mathbb{R}^2 \to \mathbb{R}^2$, $\boldsymbol{f}(t, \boldsymbol{x}) = (f_x(t, \boldsymbol{x}), f_y(t, \boldsymbol{x}))^t = (\alpha(1-r^2)x - y, x + \alpha(1-r^2)y)^t$
- $A = \begin{pmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{pmatrix} \in \mathcal{M}_{2\times 2}(\mathbb{R}) \approx \mathbb{R}^{n^2}$
- $D_{\boldsymbol{x}}\boldsymbol{f}(t, \boldsymbol{x}) = \begin{pmatrix} \frac{\partial f_x}{\partial x}(t, \boldsymbol{x}) & \frac{\partial f_x}{\partial y}(t, \boldsymbol{x}) \\ \frac{\partial f_y}{\partial x}(t, \boldsymbol{x}) & \frac{\partial f_y}{\partial y}(t, \boldsymbol{x}) \end{pmatrix}$
- $t_0 = 0$ i $\boldsymbol{x}_0 = (x(0), y(0)) = (1, 0)$
- I_2 és la matriu identitat 2×2

Sabem que la solució teòrica d'aquest problema és $(\boldsymbol{\phi}(t; t_0, \boldsymbol{x}_0), D_{\boldsymbol{x}}\boldsymbol{\phi}(t; t_0, \boldsymbol{x}_0))$, on $\boldsymbol{\phi}(t; t_0, \boldsymbol{x}_0)$ és el flux del sistema (7).

Si aplanem la matriu A per files, podem reescriure les equacions del sistema (8)

com

$$\begin{cases} \dot{\boldsymbol{X}} = \boldsymbol{F}(t, \boldsymbol{X}) \\ \boldsymbol{X}(t_0) = (\boldsymbol{x}_0, 1, 0, 0, 1) \end{cases} \tag{9}$$

on $\boldsymbol{X} = (x\,,\, y\,,\, a_{0,0}\,,\, a_{0,1}\,,\, a_{1,0}\,,\, a_{1,1})^t$ i $\boldsymbol{F}\colon \mathbb{R} \times \mathbb{R}^6 \to \mathbb{R}^6$ es la funció definida per:

$$\boldsymbol{F}(t, \boldsymbol{X}) = (F_1(t, \boldsymbol{X}), F_2(t, \boldsymbol{X}), F_3(t, \boldsymbol{X}), F_4(t, \boldsymbol{X}), F_5(t, \boldsymbol{X}), F_6(t, \boldsymbol{X}))^t$$

on

$$\begin{aligned} F_1(t, \boldsymbol{X}) &= f_x(t, \boldsymbol{x}) \\ F_2(t, \boldsymbol{X}) &= f_y(t, \boldsymbol{x}) \\ F_3(t, \boldsymbol{X}) &= \frac{\partial f_x}{\partial x}(t, \boldsymbol{x}) \cdot a_{0,0} + \frac{\partial f_x}{\partial y}(t, \boldsymbol{x}) \cdot a_{1,0} \\ F_4(t, \boldsymbol{X}) &= \frac{\partial f_x}{\partial x}(t, \boldsymbol{x}) \cdot a_{0,1} + \frac{\partial f_x}{\partial y}(t, \boldsymbol{x}) \cdot a_{1,1} \\ F_5(t, \boldsymbol{X}) &= \frac{\partial f_y}{\partial x}(t, \boldsymbol{x}) \cdot a_{0,0} + \frac{\partial f_y}{\partial y}(t, \boldsymbol{x}) \cdot a_{1,0} \\ F_6(t, \boldsymbol{X}) &= \frac{\partial f_y}{\partial x}(t, \boldsymbol{x}) \cdot a_{0,1} + \frac{\partial f_y}{\partial y}(t, \boldsymbol{x}) \cdot a_{1,1} \end{aligned}$$

Per tant, el sistema que integrarem numèricament serà el sistema (9). Un cop l'haguem integrat, fixarem t no gaire grossa i comprovarem numèricament que, llevat dels errors d'integració numèrica, les quatre últimes coordenades de la solució obtinguda avaluada en t coincideixen amb els coeficients de la diferencial del flux del sistema (7) respecte de les condicions inicials avaluada en $(t; t_0, \boldsymbol{x}_0)$, és a dir, $D_{\boldsymbol{x}}\boldsymbol{\phi}(t; t_0, \boldsymbol{x}_0)$. Per fer aquesta comprovació, aproximarem totes les derivades parcials de $\phi = (\phi_1, \phi_2)^t$ fent servir diferències finites centrades de primer ordre, és a dir,

$$D_{\boldsymbol{x}}\boldsymbol{\phi}(t; t_0, \boldsymbol{x}) = \left(\frac{\partial \phi(t; t_0, \boldsymbol{x})}{\partial x}\ ,\ \frac{\partial \phi(t; t_0, \boldsymbol{x})}{\partial y} \right) =$$

$$= \left(\frac{\phi(t; t_0, (x+\delta, y)) - \phi(t; t_0, (x-\delta, y))}{2\delta}\ ,\ \frac{\phi(t; t_0, (x, y+\delta)) - \phi(t; t_0, (x, y-\delta))}{2\delta} \right) +$$

$$+ O(\delta^2)\mathbf{1}_2$$

on $\mathbf{1}_2$ és la matriu 2×2 amb tots els coeficients iguals a 1, i comprovarem que la diferència entre aquesta matriu $(D_{\boldsymbol{x}}\boldsymbol{\phi}(t; t_0, \boldsymbol{x}_0))$ i la matriu formada per les quatre últimes coordenades de la solució del sistema (9) avaluada en t dóna lloc a una matriu amb coeficients més petits (en valor absolut) que la tolerància que fixem per la integració numèrica.

El programa que hem desenvolupat per dur a terme tot aquest procés consta de tres "mòduls". El primer mòdul són dues funcions definides al fitxer `scripts/utils/functs.py`. Una d'elles implementa el camp vectorial $\boldsymbol{f}$ del sistema (7), mentre que l'altra implementa la diferencial (matriu jacobiana) del camp $\boldsymbol{f}$.

El segon mòdul és un fitxer `scripts/utils/include_diffs.py` amb una funció `camp()` que s'encarrega d'implementar el camp $\boldsymbol{F}$ del sistema (9) a partir del camp $\boldsymbol{f}$ i la seva diferencial (es passen com a paràmetres), mitjançant el procediment d'aplanament de matrius que hem descrit abans.

El tercer mòdul és un fitxer `scripts/s2_ex1.py` que controla tot el procés. Per començar, es declaren i s'inicialitzen els valors dels paràmetres i les condicions inicials del sistema (9) així com l'instant de temps `t_pred` en què volem estimar la solució del sistema. A continuació, s'aplica la funció `camp()` a les dues funcions del primer mòdul per obtenir una funció `funct` que implementa el camp $\boldsymbol{F}$ del sistema (9). Aleshores, es crida a la funció `flux()` (mitjançant una instància de la classe `rk78`) passant les condicions inicials i tots els paràmetres i passant també la funció `funct`, per tal d'integrar numèricament el sistema (9) i estimar-ne la solució en l'instant `t_pred`. Finalment, s'aproxima la matriu $D_{\boldsymbol{x}}\boldsymbol{\phi}(\texttt{t\_pred}; t_0, \boldsymbol{x}_0)$ utilitzant l'expressió que hem descrit abans (per a un cert valor del paràmetre δ), cridant 4 cops a la funció `flux()` (amb paràmetres diferents) per dur a terme les quatre avaluacions de ϕ necessàries en aquesta aproximació de la matriu. Per últim, el programa imprimeix la diferència entre la matriu $D_{\boldsymbol{x}}\boldsymbol{\phi}(\texttt{t\_pred}; t_0, \boldsymbol{x}_0)$ i la matriu formada pels últims 4 valors de l'output de la primera crida a `flux()`.

Hem executat el fitxer `scripts/s2_ex1.py` agafant-nos els següents valors dels paràmetres:

- Paràmetres de l'EDO: $\alpha = 0.5$

- Paràmetres de la funció `flux()`:
  - `h` $= 0.001$ (pas inicial proposat)
  - (`h_min`, `h_max`) $= (0.0001, 0.005)$ (pas mínim i pas màxim permesos)
  - `tol` $= 0.001$ (tolerància)
  - `max_steps` $= 10000$ (nombre màxim de passos permesos en cada crida a `flux()`)
  - `t_pred` $= 0.5$

Mostrem a continuació la matriu de diferències obtinguda:

```
Error between the computed matrices:
[[-2.34905428e-11  4.10309564e-11]
 [-8.68821681e-12  1.92545979e-12]]
```

Observem que tots els coeficients d'aquesta matriu són bastant més petits en valor absolut que la tolerància `tol` $= 0.001$. Per tant, això demostra numèricament que, llevat dels errors d'integració numèrica, les quatre últimes coordenades de la solució del sistema (9) coincideixen amb els coeficients de la diferencial del flux del sistema (7) respecte de les condicions inicials avaluada en $\boldsymbol{x}_0$, és a dir, $D_{\boldsymbol{x}}\boldsymbol{\phi}(t; t_0, \boldsymbol{x}_0)$.

2.3 Mètode QR per sistemes sobredeterminats

La descomposició QR és un dels múltiples mètodes pràctics que es fan servir per a resoldre sistemes matricials de manera eficient. Aquesta tercera part de la pràctica consistirà en construir un algoritme de resolució de sistemes lineals per a fer possible el càlcul de maniobres, la última part de la pràctica.

No obstant, la descomposició QR es construeix d'una manera més general, prenent com a objectiu, minimitzar el valor (10):

$$\min_{x \in \mathbb{R}^n} ||Ax - b||_2 \ , \quad A \in \mathbb{R}^{m \times n},\ b \in \mathbb{R}^m \ , \quad n \leq m \tag{10}$$

A línies generals, la descomposició QR es basa en transformar A en un producte de matrius $Q \cdot R$, on $Q \in \mathbb{R}^{m \times m}$ és matriu ortogonal i $R \in \mathbb{R}^{m \times n}$ és matriu triangular

superior. Ja d'entrada, veiem que no sempre és possible obtenir aquesta descomposició (quan el rang de A és menor a n), que solament depèn de les característiques que presenta A. En qualsevol cas, quan sigui possible, donat que Q és ortogonal, es compleix $||y|| = ||Q^T \cdot y||$, i per tant, podem simplificar el nostre objectiu a l'equació (11):

$$\min_{x \in \mathbb{R}^n} ||Rx - Q^T b||_2 \ , \quad R \in \mathbb{R}^{m \times n}, \ Q^T b \in \mathbb{R}^m \tag{11}$$

Si es dona el cas, les $m - n$ últimes files de la matriu R contindran únicament zeros i podem trobar el vector x que minimitza la distància euclidiana com la solució atorgada per les n primeres files de R. A més, aquest sistema té garantida solució (ja que R és matriu triangular superior) i la podem trobar per substitució enrere, mètode molt eficient en el particular cas que tinguem la matiu del sistema triangular. Quan implementem el mètode de manera eficient, no és necessari trobar la matriu Q, sinó que apliquem transformacions per la dreta a A i b fins obtenir R i $Q^T b$ respectivament. Aquestes transformacions s'anomenen de HouseHolder, i tot i que no explicarem el desenvolupament teòric de l'algoritme, exposarem els passos que hem fet servir, de manera optimitzada, per realitzar les transformacions.

Exercici 1

Tal com esperàvem, en l'exercici 1 ens demanen implementar el mètode QR. Com que fem ús del llenguatge Python que gaudeix de llibreries que permeten treballar directament amb vectors i matrius, no seguirem ni la rutina esmentada ni l'estructura de la funció, però sí que mantindrem la nomenclatura de les variables, que apareix en el document de la pràctica.

Tal com acabem de dir, podem treballar de manera matricial, fet que a part de millorar l'eficiència, simplifica el procediment, de manera que el procediment teòric i pràctic són molt semblants.

A continuació, exposem el procediment a realitzar. Prenem $A^{(0)} = A$, $b^{(0)} = b$, que en aplicar transformacions per la dreta, obtindrem les parelles

$$(A^{(0)} = A, b^{(0)} = b), \ (A^{(1)}, b^{(1)}), \ldots, (A^{(n)} = R, b^{(n)} = Q^T b)$$

Si el rang de A és no menor a n, l'algoritme ens garanteix que podem expressar $A^{(k)}$ i $b^{(k)}$ amb $k \in \{1, \ldots, n\}$ com:

$$A^{(k)} = \left(\begin{array}{c|c} T^{(k)} & M^{(k)} \\ \hline 0 & \widetilde{A}^{(k)} \end{array}\right) \ , \quad b^{(k)} = \left(\begin{array}{c} c^{(k)} \\ \hline \widetilde{d}^{(k)} \end{array}\right)$$

on:

- $T^{(k)} \in \mathbb{R}^{k \times k}$ és matriu triangular superior.
- $M^{(k)} \in \mathbb{R}^{k \times (n-k)}$
- $\widetilde{A}^{(k)} \in \mathbb{R}^{(m-k) \times (n-k)}$
- $c^{(k)} \in \mathbb{R}^{k}$
- $\widetilde{d}^{(k)} \in \mathbb{R}^{(m-k)}$

El fet clau és que podem relacionar dues parelles consecutives com:

$$A^{(k+1)} = \left(\begin{array}{c|c} I_k & 0 \\ \hline 0 & \widetilde{P}^{(k)} \end{array}\right) \cdot A^{(k)} = \left(\begin{array}{c|c} T^{(k)} & M^{(k)} \\ \hline 0 & \widetilde{P}^{(k)} \cdot \widetilde{A}^{(k)} \end{array}\right)$$
$$b^{(k+1)} = \left(\begin{array}{c|c} I_k & 0 \\ \hline 0 & \widetilde{P}^{(k)} \end{array}\right) \cdot b^{(k)} = \left(\begin{array}{c} c^{(k)} \\ \hline \widetilde{P}^{(k)} \cdot \widetilde{d}^{(k)} \end{array}\right)$$

on $\widetilde{P}^{(k)} \in \mathbb{R}^{(m-k) \times (n-k)}$ i la podem calcular mitjançant el següent procediment:

1. Calculem la norma euclidiana de la primera columna del bloc $\widetilde{A}^{(k)}$:

$$\gamma = ||(a^{(k)}_{k+1,k+1}, \ldots, a^{(k)}_{m,k+1})||_2$$

2. Construïm un vector $\widetilde{u}^{(k)}$ transformant el vector del primer pas:

$$\widetilde{u}^{(k)} = \left(a^{(k)}_{k+1,k+1} + sig(a^{(k)}_{k+1,k+1})\gamma, a^{(k)}_{k+2,k+1}, \ldots, a^{(k)}_{m,k+1}\right)$$

3. Ja tenim els elements necessaris per construir $\widetilde{P}^{(k)}$:

$$\widetilde{P}^{(k)} = I_{m-k} - \frac{2}{\widetilde{u}^{(k)T}\widetilde{u}^{(k)}} \widetilde{u}^{(k)}\widetilde{u}^{(k)T}$$

D'aquesta manera, si en cap iteració obtenim un vector $\widetilde{u}^{(k)}$ nul, podem obtenir les matrius R i $Q^T b$ i trobar x resolent el sistema format per les primeres n files de R i $Q^T b$ mitjançant la substitució enrere.

Amb els conceptes desenvolupats, ara expliquem el procediment que hem emprat. Hem creat el fitxer `scripts/utils/system_solver.py` que conté tres funcions essencials, `qrres()`, que fa la descomposició QR per tal d'obtenir R i $Q^T b$, `solve_qr()`, funció que fa la substitució enrere donada ja la descomposició QR i finalment, la funció `solve_system()` que aplica les dues anteriors funcions per trobar la solució al problema (10).

- Comencem explicant què es fa en `qrres()`. Tal com hem explicat en el desenvolupament teòric, haurem de realitzar n iteracions modificant la matriu A i el vector b. No obstant, no volem sobreescriure les matrius que originalment ens donen i per tant, en farem una còpia, que guardarem a les variables `A_k, b_k`. A més, la funció demana els arguments `tol`, que ens dona un criteri per decidir si un vector amb components quasi nul·les és considerat nul, i `count_time`, que per defecte pren el valor `False`, i s'utilitza només en els següents apartats.

  Així, creem un iterador, on per a cada valor de $k \in \{1, \ldots, n\}$ calculem γ, $\widetilde{u}^{(k)}$ i $\widetilde{P}^{(k)}$ de manera vectoritzada, sempre i quan la norma euclidiana de $\widetilde{u}^{(k)}$ sigui major a la tolerància que rebem com argument de la funció. Si $\widetilde{u}^{(k)}$ és considerat vector nul, parem el procés i retornem els valors actuals de `A_k` i `b_k`, junt amb un booleà `success = False`. En cas contrari, realitzem el producte matricial entre $\widetilde{P}^{(k)}$ i $\widetilde{A}^{(k)}$ i $\widetilde{d}^{(k)}$ actualitzem `A_k` i `b_k`. Un cop realitzades les n iteracions, retornem `success = True` junt amb `A_k` i `b_k` que ara prenen els valors de R i $Q^T b$.

- Ara desenvolupem la funció `solve_qr()`. Com que estim partint que tenim un sistema com (11), hem de comprovar que realment els arguments d'entrada són correctes, és a dir, que $m \geq n$, $Q^T b \in \mathbb{R}^m$, i que R és matriu triangular superior a $\mathbb{R}^{m \times n}$. Aquestes verificacions són opcionals ja que costen temps. A continuació, inicialitzem un vector de solucions a zero (per tant, de dimensió n), que anirem actualitzant començant per la component n a mesura que fem

la substitució enrere. Així, coneixent $x_{k+1}, \ldots, x_n$, podem trobar x_k realitzant:

$$\begin{aligned} x_k &= (Q^T b)_k - \langle (r_{k,k+1}, \ldots, r_{k,n}), (x_{k+1}, \ldots, x_n) \rangle / r_{k,k} \\ &= (Q^T b)_k - \langle (r_{k,1}, \ldots, r_{k,n}), (x_1, \ldots, x_n) \rangle / r_{k,k} \end{aligned}$$

on l'última igualtat l'hem pogut aplicar ja que $x_1, \ldots, x_k$ estan inicialitzats a zero. Realitzant el procés per $k \in \{n \ldots, 1\}$, trobem la solució, que la retornem.

- Finalment, ja només resta explicar la funció `system_solver()`. Com arguments, té la matriu A i vector b del sistema (10), i `tol`, que s'utilitza com a criteri per decidir si un valor o un vector és nul. Aquesta funció és molt simple: primer crida la funció `qrres()`, que donat A i b et retorna R i $Q^T b$, i si és possible la descomposició QR. En cas que no s'hagi pogut fer, la funció no retorna res i mostra un missatge per consola. En cas contrari, s'utilitza la funció `solve_qr()`, que a partir de R i $Q^T b$, trobem la solució que és el que retornem a la funció `system_solver()`.

Amb aquest procediment, hem completat la tasca que se'ns demana a l'exercici 1 (`qrres()`) i, hem afegit les funcions `solve_qr()` i `system()` que permet l'usuari utilitzar el mètode QR de manera més senzilla.

Exercici 2

Amb el mètode QR construït, hem de verificar el funcionament del programa mitjançant l'exemple donat:

$$A = \begin{pmatrix} 0 & -4 \\ 0 & 0 \\ 5 & -2 \end{pmatrix} \quad , \quad b = \begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix}$$

Hem creat un fitxer `scripts/s3_ex2.py`, que importa la funció `system_solver()` del fitxer `scripts/utils/system_solver.py`. Inicialitzem les variables `A`, `b` amb els valors de l'enunciat, apliquem la funció `system_solver(A, b)` i mostrem tant la solució obtinguda com la solució real, $(0.3, -0.25)$. A continuació, mostrem les matrius i vectors intermedis i finals:

Valors intermedis

```
k
0
A_k
array([[ 0., -4.],
       [ 0.,  0.],
       [ 5., -2.]])
b_k
array([1., 3., 2.])
u_k
array([5., 0., 5.])
```

```
k
1
A_k
array([[-5.,  2.],
       [ 0.,  0.],
       [ 0.,  4.]])
b_k
array([-2.,  3., -1.])
u_k
array([4., 4.])
```

Valors finals

```
Rst
array([[-5.,  2.],
       [ 0., -4.],
       [ 0.,  0.]])
Qtb
array([-2.,  1., -3.])
```

```
Check solution:
Real solution : [ 0.3  -0.25]
Found solution: [ 0.3  -0.25]
```

Hem mostrat tots els valors intermedis i finals, que coincideixen amb els que ofereix l'enunciat de la pràctica i coincideixen.

Exercici 3

En aquest exercici, ens demanen estimar la precisió del mètode sobre la resposta i per fer-ho, ens demanen calcular l'error de determinació:

$$\max_{1 \leq i \leq n} |x_i - 1|$$

sobre sistemes $Ax = b$, amb A aleatòria i b construïda tal que $x = (1, \ldots, 1)$ sigui solució al sistema. Hem resolt l'exercici en el fitxer `scripts/s3_ex3.py`. Altre cop, hem importat la funció `solve_system()` del fitxer `scripts/utils/system_solver.py`,

hem generat 10 valors aleatoris entre 3 i 50 de n, i per a cada n, generem A aleatòriament, trobem b tal que $x = (1, \ldots, 1)$ és solució del sistema $Ax = b$ i calculem la solució mitjançant QR. Finalment, calculem i mostrem l'error de determinació. El resultat depèn de la pseudo-aleatorietat però en tot cas, hem executat un cop i hem obtingut la Figura 6. Observem que els valors de n que apareixen es troben entre 9 i 49 i en qualsevol cas, l'error de determinació és com a màxim, de l'ordre de 10^{-13}, que fa el mètode QR implementat prou precís.

Exercici 4

En el segon exercici, hem verificat amb un exemple concret el funcionament del mètode. En el tercer exercici, hem estudiat la precisió del mètode. Finalment, estudiem l'eficiència de l'algoritme implementat. Específicament, ens centrem estrictament en la descomposició QR i l'objectiu és estudiar el temps emprat en el procés en funció de la dimensió del sistema.

Tal com s'exposa en el document de la pràctica, el nombre d'operacions emprades és de l'ordre de n^3, de fet, $4n^3/3 + O(n^2)$. Si es pren la suposició que el nombre d'operacions és proporcional al temps emprat per a realitzar-les, llavors, el temps emprat en realitzar la descomposició també ha de ser de l'ordre de n^3 i la divisió entre el temps emprat i $4n^3/3$ hauria de resultar en una funció prou semblant a una funció constant amb constant diferent a zero per n mínimament gran, per exemple, $n \geq 30$.

Hem creat el fitxer `scripts/s3_ex4.py`, que de manera simplificada, genera sistemes lineals aleatoris de dimensió $n \times n$ entre el rang escollit (per defecte $40 \leq n < 200$), calcula el temps emprat en fer la descomposició per a cadascun, i guarda el temps en el fitxer `outputs/section3/ex4/ratio_output/ratio.txt`.

Més concretament, importem la funció `qrres()` del fitxer `scripts/utils/system_solver()`, i ara és quan hem d'introduir l'argument `count_time = True`. En aquest cas, dins de la funció només s'aplicaran els canvis sobre A i ignorarem les transformacions aplicades a b. En l'inici de les transformacions, amb la llibreria `time`, calcularem els valors `start_time` i `time_end`, i retornarem la resta, que és el temps emprat en fer estrictament la descomposició QR.

```
Max element error in each of 10 random systems of random dimension:

n =  9
Max element error | x_i - 1|:
5.329070518200751e-15

n =  27
Max element error | x_i - 1|:
8.328893130737924e-13

n =  26
Max element error | x_i - 1|:
2.220446049250313e-14

n =  22
Max element error | x_i - 1|:
2.930988785010413e-14

n =  19
Max element error | x_i - 1|:
8.881784197001252e-15

n =  49
Max element error | x_i - 1|:
2.90878432451791e-14

n =  18
Max element error | x_i - 1|:
7.105427357601002e-15

n =  16
Max element error | x_i - 1|:
3.9968028886505635e-15

n =  15
Max element error | x_i - 1|:
3.042011087472929e-14

n =  39
Max element error | x_i - 1|:
5.88418203051333e-14
```

Figura 6: Resposta al executar el fitxer `scripts/s3_ex3.py`.

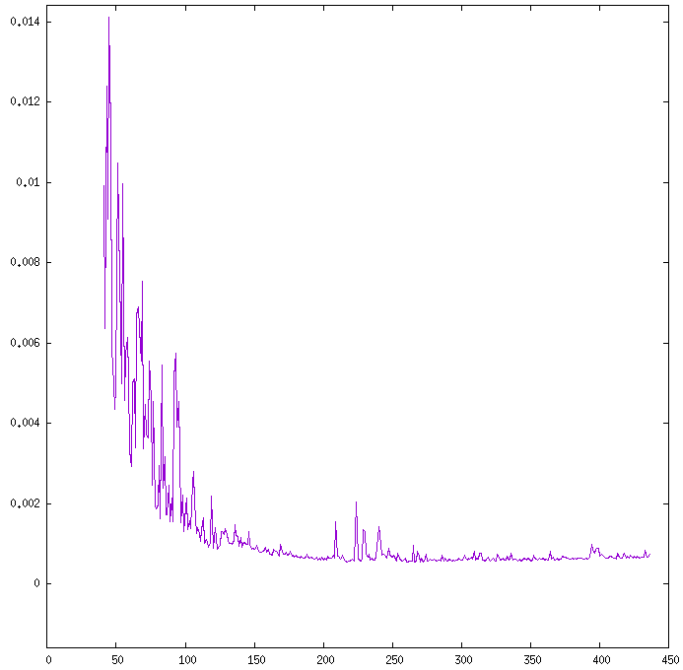


Figura 7: Eficiència del model. Comparem nombre d'operacions mínimes i temps emprat.

Llavors, tal com hem dit, calcularem el temps emprat en fer el QR per matrius de dimensió $n \in \{40, \ldots, 199\}$, i aquestes matrius les generem de manera aleatòria. A continuació, mostrem una gràfica de $\left(n, \frac{20000 \cdot t_n}{4n^3/3}\right)$ on t_n és el temps emprat en el sistema lineal de dimensió $n \times n$, prenent el rang de dimensions com $\{40, 439\}$, Figura 7. Veiem que per n major a 200, es comporta aproximadament com una funció constant, i que per tant, sota la suposició que el temps emprat és proporcional al nombre d'operacions realitzades, l'algoritme desenvolupat és competent.

2.4 Càlcul de maniobres

Aquesta secció és la última i és l'objectiu real de la pràctica, al qual hem pogut arribar gràcies als anteriors apartats guiats. En l'introducció de la pràctica, està explicat en detall l'objectiu concret i el procediment teòric a seguir i per tant, ens centrarem específicament a resoldre els apartats guiats d'aquesta última part de la pràctica.

Exercici 1

Si considerem les funcions G i DG introduïdes en les equacions (5, 6), ens demanen crear una funció `cmani_gdg` que donat el temps Δt, la posició inicial, final, i els increments de velocitat, calculi G i DG a temps Δt. No entrarem en detall en la resolució ja que el procediment és força farragós i està explícitament comentat en el codi. No obstant, explicarem resumidament el desenvolupament.

Hem creat un fitxer `scripts/utils/maneuvers.py` que conté la classe `Maneuvers` on hi incloem totes les utilitats. La primera utilitat, és el mètode `cmani_gdg` que és la resolució d'aquest exercici. La classe `Maneuvers` té com a arguments els paràmetres d'integració numèrica que es faran servir en qualsevol de les utilitats de la classe, en particular, del mètode `cmani_gdg`. Llavors, el mètode `cmani_gdg` té com a paràmetres Δt, la posició inicial, la posició final, els increments de velocitat, i les funcions F i DF a partir de les quals, podem generar el flux i la variacional associada. Per últim, té un paràmetre de tipus booleà que serveix per escollir si l'usuari vol fer les comprovacions necessàries sobre els inputs. El mètode retorna, si és possible, un diccionari que conté l'avaluació de G i DG en temps Δt.

Exercici 2

En aquest segon exercici, ens demanen construir una funció de nom `cmani()` que trobi arrels de G mitjançant el mètode de Newton. Aquesta rutina està implementada en el mateix fitxer `scripts/utils/maneuvers.py`, i és el segon mètode de la classe `Maneuvers`. Gràcies a la construcció del mètode `cmani_gdg`, ara es simplifica l'explicació del procediment emprat.

Com a paràmetres d'entrada, escollim altre cop Δt, la posició inicial, la posició final, els increments de velocitat inicials, i les funcions F i DF. A més demanem un nombre màxim d'iteracions que pot realitzar el mètode de Newton, en cas que en cap iteració anterior no s'hagi pogut acotar la norma euclidiana de G per la tolerància proporcionada en la inicialització de la classe.

Així, per a cada iteració del mètode de Newton, avaluem G i DG, que es guarden en la variable `output`. Si h és funció a $\mathbb{R}^n$ i $x^{(0)}$ és llavor, el mètode de Newton

vé donat per:

$$x^{(k+1)} = x^{(k)} - Dh(x^{(k)})^{-1} \cdot h(x^{(k)})$$

i per tant, $x^{(k+1)} - x^{(k)}$ és solució al sistema $Dh(x^{(k)}) \cdot x = -h(x^{(k)})$. En el nostre cas, h és G, i com veiem, no és necessari invertir la matriu DG sinó que podem trobar els increments de velocitat de la iteració $k+1$, $\Delta v^{(k+1)}$, mitjançant la resolució del sistema $DG(\Delta v^{(k)})x = -G(\Delta v^{(k)})$. Així, sumem $\Delta v^{(k)}$ a la solució del sistema per obtenir $\Delta v^{(k+1)}$. Per resoldre el sistema, fem servir el mètode QR que hem desenvolupat a l'apartat 3 de la pràctica.

D'aquesta manera, hem construït el procés iteratiu de Newton que parem si l'avaluació de G en $\Delta v^{(k)}$ és prou petita o si excedim el nombre màxim d'iteracions. Sigui quin sigui el motiu de parada, retornem els increments calculats a la última iteració i printem un missatge mostrant si els increments trobats són considerats una arrel de G.

Exercici 3

Aquest exercici consisteix bàsicament a comprovar que els 2 exercicis anteriors estan realitzats correctament, aplicant-los al problema del pèndol. Partint de la posició inicial $(1, 0)$, si prenem la posició final $(0, -0.95885108)$, hem de trobar els increments de velocitat tals que en $\Delta t = \pi/2$ ens trobem a la posició final. Hem de realitzar el procés iterat de Newton prenent la llavor que els increments de velocitat són nuls. Així, hem creat el fitxer `scripts/s4_ex3.py`, on es realitza la tasca.

Importem la classe `Maneuvers` del fitxer `scripts/utils/maneuvers.py` i la funció pèndol i diferencial del pèndol del fitxer `scripts/utils/functs.py`. Declarem les variables necessàries prèviament anomenades, també prenem els paràmetres d'integració numèrica, prenem una instància de la classe `Maneuvers`, i cridem el mètode `cmani`, que ens troba els increments buscats. A continuació, mostrem els valors obtinguts:

```
Success: True
[[-0.09269816]
 [ 0.00620787]]
```

que observem que coincideixen amb els increments proporcionats al document de la pràctica.

Exercici 4

Aquest és l'últim exercici de la pràctica i és l'aplicació de l'exercici 2 d'aquesta quarta part de la pràctica al problema dels 3 cossos. De la mateixa manera que en l'anterior exercici, ens demanen trobar els increments, ara però, prenent cada increment a $\mathbb{R}^3$ per al problema dels tres cossos.

Hem creat el fitxer `scripts/s4_ex4.py` on desenvolupem l'exercici. La posició inicial, la posició final i Δt les extraiem del fitxer `outputs/section4/ex4/condicions_inicials/cmani_rtbp.inp`. Per altra banda, inicialitzem la llavor dels increments de velocitat a vectors nuls. El procediment és exactament el mateix que hem realitzat en l'anterior exercici, ara guardem els increments en el fitxer `outputs/section4/ex4/boosts/boosts.txt`. Prenent les condicions inicials del fitxer `cmani_rtbp.inp`, els increments trobats són (reduint els decimals per truncament):

$$\begin{aligned}\Delta v = \big((&-8.3261e-06, -1.2856e-05, -1.5458e-06),\\ &(-5.9891e-06, -1.0711e-05, -1.8092e-06)\big)\end{aligned} \tag{12}$$

Havent trobat els increments, resten dues tasques. La primera, és verificar que amb els increments trobats, a temps Δt, s'assoleix la posició final. La segona, és fer una representació gràfica de la trajectòria del satèl·lit.

Per realitzar les dues tasques, hem creat un fitxer addicional, `scripts/s4_ex4_bonus_track.py`, que a partir de l'integrador numèric rk78, construït a l'apartat 1 de la pràctica, calculem la trajectòria del satèl·lit, tenint en compte els increments de velocitat. Per això, necessitem extreure únicament els increments de velocitat de `outputs/section4/ex4/boosts/boosts.txt` i la posició inicial i Δt de `outputs/section4/ex4/condicions_inicials/cmani_rtbp.inp`. D'aquesta manera, usant el mètode `flux_multistep()` de la classe `rk78`, calcula les `num_evals = 1000` posicions intermèdies que pren el satèl·lit, que guardem en el fitxer `outputs/section4/ex4/orbites_output/orbites.txt`. A més, generem el fitxer `outputs/section4/ex4/orbites_output/points.txt` que conté les posicions de les dues masses principals.

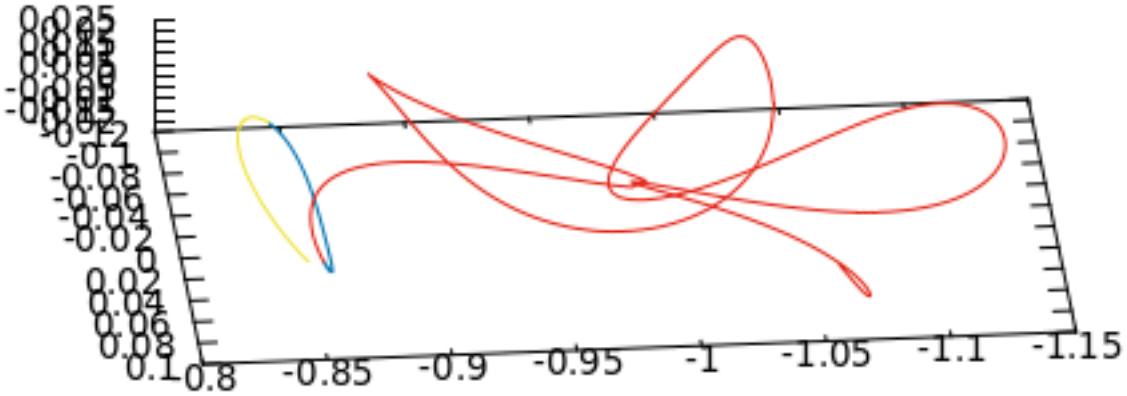


Figura 8: Trajectòria del satèl·lit.

Si obrim el penúltim fitxer mencionat, observem que a $\Delta t = \pi/2$ (línia 2004 del fitxer), el satèl·lit es troba a la posició (havent reduit decimals per truncament):

$$\begin{aligned} r'_f = (&-8.518851062430521370 \cdot 10^{-1}, \\ &6.132143191603947607 \cdot 10^{-2}, \\ &5.120725159302753080 \cdot 10^{-4}) \end{aligned}$$

En canvi, la posició final esperada és:

$$\begin{aligned} r_f = (&-8.518851062423166 \cdot 10^{-1}, \\ &6.132143191581388 \cdot 10^{-2}, \\ &5.120725158794939 \cdot 10^{-4}) \end{aligned}$$

Per obtenir la posició r'_f, hem utilitzat els increments (12), i llevat d'errors comesos en la integració numèrica, es compleix per construcció de G i del mètode de Newton que $||r'_f - r_f||^2 < 10^{-15}$. Per tant, verifiquem que els increments (12) estan correctament calculats i són arrels de G.

Per a la segona tasca, grafiquem la trajectòria del satèl·lit continguda en el fitxer `outputs/section4/ex4/orbites_output/orbites.txt` i obtenim la Figura 8. A la Figura, a temps inicial, el satèl·lit es troba a l'extrem de color groc, a temps $\pi/4$, quan canvia de color groc a blau, rep el segon increment de velocitat fins arribar

a temps $\pi/2$ que arribem a la posició final, on la corba canvia de color de blau a vermell. Finalment, la continuació de la corba, és la trajectòria que segueix aquest satèl·lit de manera lliure per temps superiors a $\pi/2$, i observem que justament havent pres $\mu \approx 0.01215$, hi ha una massa principal a la posició $(\mu - 1, 0, 0)$, que explica la trajectòria erràtica del satèl·lit.

3 Bonus track: Una aplicació del mètode de Newton per a la detecció d'òrbites periòdiques en sistemes diferencials autònoms

En aquesta secció presentem un treball voluntari que hem realitzat per tal de poder aprofundir en alguns aspectes de l'assignatura i, alhora, poder relacionar els mètodes apresos amb altres disciplines de les matemàtiques. El desenvolupament teòric que realitzarem a continuació el durem a terme a $\mathbb{R}^3$, tot i que serà fàcilment generalitzable[3] a dimensions superiors, tal i com quedarà clar durant el procés.

Considerem un sistema diferencial autònom a $\mathbb{R}^3$ arbitrari. Diguem:

$$x' = f(x) \tag{13}$$

on suposem per simplicitat que $x \in \mathbb{R}^3$, tot i que no té importància. Cal suposar també que f té la suficient regularitat per tal que tot el que fem a continuació sigui vàlid. Denotem per $\varphi(t, x_0, y_0, z_0)$ el flux de (13), és a dir, per a cada (x_0, y_0, z_0) fixat, la funció φ [4] és solució de (13) i a més verifica que $\varphi(0, x_0, y_0, z_0) = (x_0, y_0, z_0)$. Necessitarem un xic més de notació. Posem:

$$\varphi(t, x_0, y_0, z_0) = (y_1(t), y_2(t), y_3(t))$$

Per tant, tindrem que: $\varphi(t, x_0, y_0, z_0) = (y_1(t), y_2(t), y_3(t)) = (x_0, y_0, z_0)$Cal tenir en compte que les funcions $y_i(t)$ depenen de x_0, y_0 i z_0 [5], però ometem aquest fet a la notació per tal de no sobrecarregar-la. No obstant això, és molt important

[3] De fet, completament anàleg.
[4] Vista com a funció de t, i definida en un cert domini que conté el 0.
[5] Podríem fer exactament el mateix fixant $y_0 \in \mathbb{R}$ o $z_0 \in \mathbb{R}$.

que el lector ho tingui en compte durant tota la redacció. Si fixem $z_0 \in \mathbb{R}$, podem considerar la següent funció:

$$(t, a, c) \xrightarrow{f} \varphi(t, a, c, z_0) - (a, c, z_0) = (y_1(t) - a, y_2(t) - c, y_3(t) - z_0)$$

Remarquem que la funció f depèn t, a i c però **no** de z_0 (que és un valor fixat). Conseqüentment, les funcions $y_i(t)$ dependran també de t, a i c però **no** de z_0. Observem que si trobem un punt (T, A, C) tal que $f(T, A, C) = 0$ amb $T \neq 0$ aleshores l'òrbita:

$$t \longrightarrow \varphi(t, A, C, z_0)$$

és una òrbita periòdica. Ara, assumim que la funció f té un zero $P := (T, A, C)$ amb $T \neq 0$. Aleshores, podem usar el *Mètode de Newton* 3−dimensional per tal d'aproximar-lo [6]. Tal i com vam veure a teoria, el mètode ve descrit de la següent manera:

$$I_n := \begin{bmatrix} T^n \\ A^n \\ C^n \end{bmatrix} = \begin{bmatrix} T^{n-1} \\ A^{n-1} \\ C^{n-1} \end{bmatrix} - Jf(T^{n-1}, A^{n-1}, C^{n-1})^{-1} \left(f(T^{n-1}, A^{n-1}, C^{n-1})\right)^T$$

on:

$$I_0 := \begin{bmatrix} T^0 \\ A^0 \\ C^0 \end{bmatrix} \approx \begin{bmatrix} T \\ A \\ C \end{bmatrix}$$

és una aproximació inicial. És ben conegut que si I_0 és suficientment proper a P aleshores $I_n \to P$ quan $n \to \infty$. Ara, el problema és, evidentment, calcular $Jf(T^{n-1}, A^{n-1}, C^{n-1})$. Tenim que:

$$Jf(T^{n-1}, A^{n-1}, C^{n-1}) = \begin{bmatrix} \frac{\partial y_1}{\partial t}(T^{n-1}) & \frac{\partial y_1}{\partial a}(T^{n-1}) - 1 & \frac{\partial y_1}{\partial c}(T^{n-1}) \\ \frac{\partial y_2}{\partial t}(T^{n-1}) & \frac{\partial y_2}{\partial a}(T^{n-1}) & \frac{\partial y_2}{\partial c}(T^{n-1}) - 1 \\ \frac{\partial y_3}{\partial t}(T^{n-1}) & \frac{\partial y_3}{\partial a}(T^{n-1}) & \frac{\partial y_3}{\partial c}(T^{n-1}) \end{bmatrix}$$

Ara, denotem:

$$y_{i,j}(t) = \frac{\partial y_i}{\partial j}(t) \quad and \quad y'_{i,j}(t) = \frac{\partial y_{i,j}}{\partial t}(t)$$

[6] També podríem usar el *mètode de Broyden.*

on $i = 1, 2, 3$, $j = a, c$ i

$$y_i'(t) = \frac{\partial y_i}{\partial t}(t)$$

per $i = 1, 2, 3$. Gràcies a les *equacions variacionals* sabem que les funcions anteriors satisfan la següent EDO (a $\mathbb{R}^9$!!!)[7]:

$$\begin{cases} y_1'(t) = P(y_1(t), y_2(t), y_3(t)) \\ y_2'(t) = Q(y_1(t), y_2(t), y_3(t)) \\ y_3'(t) = R(y_1(t), y_2(t), y_3(t)) \\ \begin{pmatrix} y_{1a}'(t) \\ y_{2a}'(t) \\ y_{3a}'(t) \end{pmatrix} = DX(y_1(t), y_2(t), y_3(t)) \begin{pmatrix} y_{1a}(t) \\ y_{2a}(t) \\ y_{3a}(t) \end{pmatrix} \\ \begin{pmatrix} y_{1c}'(t) \\ y_{2c}'(t) \\ y_{3c}'(t) \end{pmatrix} = DX(y_1(t), y_2(t), y_3(t)) \begin{pmatrix} y_{1c}(t) \\ y_{2c}(t) \\ y_{3c}(t) \end{pmatrix} \\ y_1(0) = A^{n-1}, y_2(0) = C^{n-1}, y_3(0) = z_0 \\ y_{1a}(0) = 1, y_{2a}(0) = 0, y_{3a}(0) = 0 \\ y_{1c}(0) = 0, y_{2c}(0) = 1, y_{3c}(0) = 0 \end{cases} \tag{14}$$

on hem denotat $X = (P(x, y, z), Q(x, y, z), R(x, y, z)) = f(x, y, z)$, on f és la funció del sistema (13). Per altra banda, notem que:

$$\left(f(T^{n-1}, A^{n-1}, C^{n-1})\right)^T = \begin{bmatrix} y_1(T^{n-1}) - A^{n-1} \\ y_2(T^{n-1}) - C^{n-1} \\ y_3(T^{n-1}) - z_0 \end{bmatrix}$$

En conclusió, la iteració del mètode de Newton ve donada per:

$$\begin{bmatrix} T^n \\ A^n \\ B^n \end{bmatrix} = \begin{bmatrix} T^{n-1} \\ A^{n-1} \\ B^{n-1} \end{bmatrix} - \begin{bmatrix} \frac{\partial y_1}{\partial t}(T^{n-1}) & \frac{\partial y_1}{\partial a}(T^{n-1}) - 1 & \frac{\partial y_1}{\partial c}(T^{n-1}) \\ \frac{\partial y_2}{\partial t}(T^{n-1}) & \frac{\partial y_2}{\partial a}(T^{n-1}) & \frac{\partial y_2}{\partial c}(T^{n-1}) - 1 \\ \frac{\partial y_3}{\partial t}(T^{n-1}) & \frac{\partial y_3}{\partial a}(T^{n-1}) & \frac{\partial y_3}{\partial c}(T^{n-1}) \end{bmatrix}^{-1} \begin{bmatrix} y_1(T^{n-1}) - A^{n-1} \\ y_2(T^{n-1}) - C^{n-1} \\ y_3(T^{n-1}) - z_0 \end{bmatrix} \tag{15}$$

[7] Si volguéssim buscar òrbites periòdiques a $\mathbb{R}^n$ amb $n \geq 4$ aleshores el sistema que ens quedarà anirà augmentant de dimensió a mesura que augmentem n.

Finalment, remarquem que:

$$\frac{\partial y_1}{\partial t}(T^{n-1}) = P(y_1(T^{n-1}), y_2(T^{n-1}), y_3(T^{n-1}))$$

$$\frac{\partial y_2}{\partial t}(T^{n-1}) = Q(y_1(T^{n-1}), y_2(T^{n-1}), y_3(T^{n-1}))$$

i

$$\frac{\partial y_3}{\partial t}(T^{n-1}) = R(y_1(T^{n-1}), y_2(T^{n-1}), y_3(T^{n-1}))$$

És molt important que tinguem aquest fet en compte ja que quan l'ordinador resol el sistema (14) calcula $y_1(2)$, $y_2(2)$ i $y_3(t)$ però no les seves derivades, fet que fa fonamental la relació anterior. Remarquem també que, tal i com hem comentat anteriorment, el procés que acabem de descriure pot ser modificat fixant x_0 o y_0 a f en comptes de z_0. Això dependrà del que ens vagi millor en cada sistema en particular. Per altra banda, el procés que acabem de descriure és computacionalment factible!. Nosaltres l'hem programat usant el software *Maple* i veurem els resultats que hem obtingut en breu. L'únic problema del mètode és que necessitem conèixer una condició inicial I_0 suficientment propera a P. I perquè? Doncs perquè el mètode de Newton només garanteix la convergència en cas que la condició inicial sigui suficientment propera a P. En alguns casos, podria que ser que no necessitéssim aproximacions inicials gaire precises ja que el mètode convergeix amb facilitat si ens allunyem de P. No obstant això, hi ha alguns sistemes diferencials que són molt sensibles respecte les condicions inicials[8], i per tant cal anar en compte. Així doncs, la pregunta és: Com trobem una aproximació inicial prou bona? I la resposta és en general ens hem d'espavilar com puguem. Moltes vegades ajudant-nos de *Maple* és relativament fàcil trobar condicions inicials corresponents a òrbites properes a òrbites periòdiques, i aquestes condicions inicials poden ser utilitzades com a llavor inicial del mètode de Newton. Després veurem que aquest mètode és molt efectiu i fàcil d'usar. Una altre manera és afegir paràmetres a (13) i buscar sistemes "propers" (en el sentit del teorema de la dependència continua respecte ens paràmetres) de manera que per algun valor dels paràmetres coneixem la localització exacte d'una òrbita periòdica (i el seu període). Llavors, podríem intentar usar un punt d'aquesta òrbita periòdica i el seu període com a llavor inicial per a la nostre iteració de Newton (el teorema de dependència continua respecte els paràmetres ens justifica

[8]Com per exemple el sistema diferencial de Lorenz, el qual estudiarem a continuació.

que això podria funcionar).

Abans de posar en pràctica el desenvolupament teòric que hem fet, ens preguntem: **Perquè usem el mètode de Newton?** Doncs perquè quan convergeix, és el millor. El mètode de Newton té dos inconvenients fonamentals: Cal escollir una condició inicial suficientment propera, i és molt costós computacionalment. No obstant això, en termes de convergència (sense tenir en compte el temps de càlcul) és el que convergeix en menys iterats, almenys genèricament. És cert que a vegades altres mètodes podrien aportar-nos convergències similars a Newton i menys esforç computacional, fet que faria factible i justificable la seva utilització. No obstant, aquest no és un fet general fet que fa que la norma general sigui: sempre que puguem, usem Newton. Una altre alternativa que hem provat per buscar cicles límit és el mètode de la secant. També ens ha funcionat, però ens dona resultats molt pitjors que els de Newton.

Passem a implementar el desenvolupament teòric descrit anteriorment en un cas particular molt interessant: el sistema diferencial de Lorenz, que és segurament el primer exemple que es va donar de caos. Així doncs, considerem el sistema d'equacions diferencials de Lorenz:

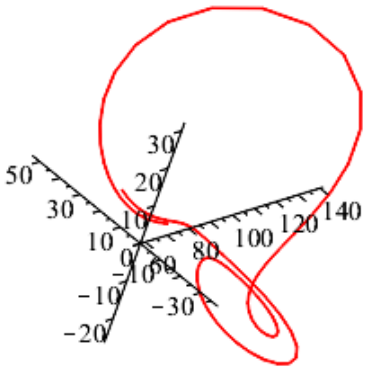
$$(x', y', z') = (\sigma(-x + y), rx - y - xz, -bz + xy) \tag{16}$$

Estudiarem dos casos:

Cas $(\sigma, r, b) = (10, 100.5, 8/3)$**.**

L'objectiu és trobar alguna òrbita periòdica de (16) amb els valors dels paràmetres fixats en $(\sigma, r, b) = (10, 100.5, 8/3)$. Jugant una mica amb el graficador de *Maple*, no costa gaire trobar condicions inicials tals que les seves òrbites periòdiques associades són molt properes a òrbites periòdiques[9]. Per exemple, si grafiquem l'orbita amb condició inicial $(0, 5, 75)$ obtenim:

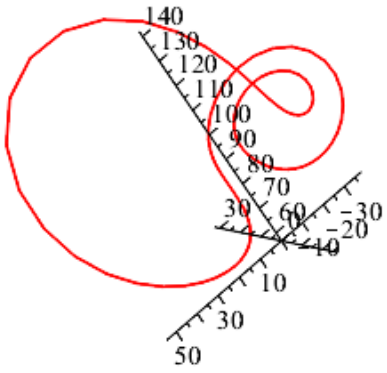
[9]Per fer-ho, un mètode útil i sovint efectiu és calcular primer els punts crítics i estudiar les òrbites periòdiques que envolten els punts crítics.



Sembla que aquesta òrbita estigui a prop d'una òrbita periòdica, tot i que clarament no ho és. Per sort, podem usar el mètode que hem descrit anteriorment per trobar una òrbita periòdica propera a l'anterior. Fixem $z_0 = 75$. L'objectiu és trobar un zero de la funció f en aquest cas particular a partir del procés que hem descrit anteriorment. A més, usarem la següent condició inicial:

$$I_0 = \begin{bmatrix} T^0 \\ A^0 \\ C^0 \end{bmatrix} = \begin{bmatrix} 1.09217 \\ 0 \\ 5 \end{bmatrix}$$

on T^0 és una aproximació del temps de retorn de l'òrbita anterior que hem trobat amb *Maple* fàcilment. Efectivament, el mètode anterior ens ha permès trobar una òrbita periòdica (una aproximació, parlant rigorosament), la qual mostrem en la següent imatge:

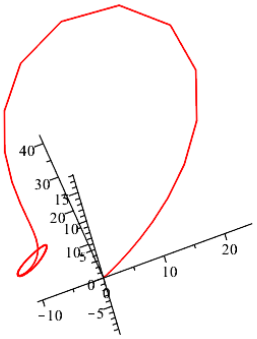


Cas $(\sigma, r, b) = (10, 25, 8/3)$**.**

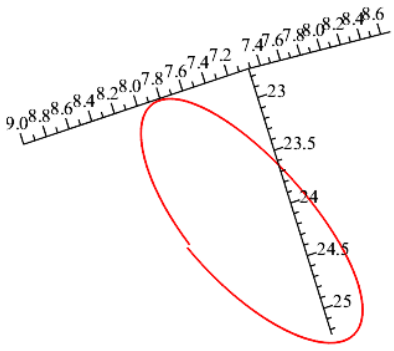
La idea és exactament la mateixa que abans. Fixem els valors dels paràmetres en $(\sigma, r, b) = (10, 25, 8/3)$. Si calculem els punts crítics de (16) per aquests valors dels paràmetres obtenim que són:

$$(0, 0, 0) \ \ (8, 8, 24) \ \ (-8, -8, 24)$$

Si grafiquem òrbites prop de l'origen ja veiem de seguida que no son gens properes a òrbites periòdiques, tal i com mostra la següent imatge:



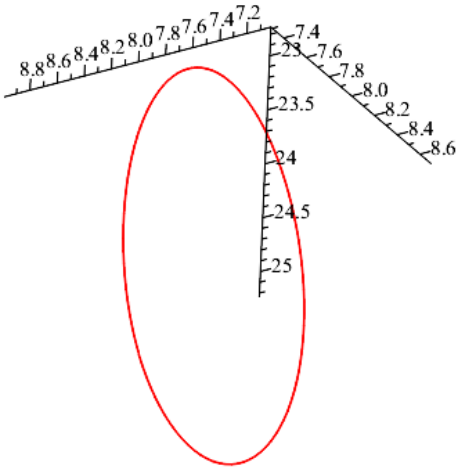
Tot i així, ja veiem que l'anterior òrbita s'embolcalla en un extrem. Això ja ens fa sospitar que al voltant d'algun dels altres punts crítics hi haurà òrbites periòdiques. Per exemple, si grafiquem òrbites periòdiques prop del punt crític $(8, 8, 24)$ ens adornarem ràpidament que les òrbites són properes a òrbites periòdiques. Si juguem una mica amb els paràmetres trobarem que, per exemple, l'òrbita associada al punt $(8.5, 9, 24)$ és molt propera a una òrbita periòdica, tal i com mostrem a la següent imatge:



A partir d'aquesta òrbita periòdica podem començar una iteració de Newton com hem fet en el cas anterior. Per tant, fixem $z_0 = 24$ i l'objectiu és trobar un zero de la funció f en aquest cas particular amb el procés que hem descrit anteriorment. Usarem la següent condició inicial:

$$I_0 = \begin{bmatrix} T^0 \\ A^0 \\ C^0 \end{bmatrix} = \begin{bmatrix} 0.65211 \\ 8.5 \\ 9 \end{bmatrix}$$

on T^0 és una aproximació del temps de retorn de l'òrbita anterior que hem trobat amb *Maple* fàcilment. Efectivament, el mètode anterior ens ha permès trobar una òrbita periòdica (una aproximació, parlant rigorosament), la qual mostrem en la següent imatge:



Utilitzant diferents condicions inicials podríem intentar trobar-ne més. Remarquem que, tal i com hem comentat abans, el sistema (16) presenta caos per alguns valors dels paràmetres, fet que força que haguem d'anar amb compte amb la tria de les condicions inicials en el mètode de Newton.

Referències

[1] F.Dumortier, J.Llibre, J.C.Artés. *Qualitative Theory of Planar Differential Systems*, Springer.

[2] J.C.Artés, J.Llibre, Dana Schlomiuk, Nicolae Vulpe. *Geometric Configurations of Singularities of Planar Polynomial Differential Systems*, Birkhäuser.

[3] M.Moskowitz, F.Paliogiannis. *Functions of Several Real Variables*, World Scientific.

[4] F.Mañosas. *Un curs d'equacions diferencials.*

[5] V.J.López. *Ecuaciones Diferenciales, cómo aprenderlas, cómo enseñarlas*, Universidad de Murcia.

[6] Stephen Lynch. *Dynamical Systems with Applications using Maple*, Birkhäuser.

[7] Carmen Chicone. *Ordinary Differential Equations with Applications*, Texts in Applied Mathematics 34, Springer.

[8] Lawrence Perko. *Differential Equations and Dynamical Systems*, Texts in Applied Mathematics 7, Springer, Third Edition.