Càlcul Numèric - **Pràctica en C**

Control discret de trajectòries mitjançant maniobres impulsives

Martín Hernán Campos Heredia

Universitat Autònoma de Barcelona

Febrer 2016

Universitat Autònoma de Barcelona

Introducció

La temàtica d'aquesta pràctica està orientada en el moviment de cossos celestes i el control de les seves trajectòries. En particular, tractem el problema d'un cos artificial (una nau, un satèl·lit) afectat per la gravetat d'altres dos cossos celestes (com bé poden ser la Terra i la Lluna) i com fer-lo arribar des d'una posició inicial fins a una posició final mitjançant dos impulsos discrets.

Més concretament, el problema a resoldre és, donades unes condicions inicials $\mathbf{r}_0, \mathbf{v}_0 \in \mathbb{R}^3$, unes condicions finals $\mathbf{r}_f, \mathbf{v}_f \in \mathbb{R}^3$ i un temps $\Delta t \in \mathbb{R}$, integrar l'equació que regeix aquest moviment (el problema restringit de 3 cossos) i trobar quins són els impulsos $\Delta\mathbf{v}_0, \Delta\mathbf{v}_1 \in \mathbb{R}^3$ -un al començament i l'altre just a $\Delta t/2$- que fan que les condicions finals de la nau siguin $\mathbf{r}_f, \mathbf{v}_f$.

Tots els codis desenvolupats es troben a la corresponent carpeta de codis i les instruccions de compilació i funcionament (entrada/sortida) de cadascun s'expliciten com a comentaris dins del mateix codi.

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

Per integrar el nostre camp, fem servir el mètode de Runge-Kutta-Felhberg d'ordres 7 i 8. Es disposa d'un fitxer `rk78.c` que conté la funció `rk78()` que duu a terme l'algoritme. En aquesta secció, creem rutines que fan servir aquesta funció per implementar el mètode i integrar alguns camps.

1-2.) Comencem pel programa `rf_pendol.c` que integra el pèndol simple

$$\begin{cases} \dot{x}(t) = y(t) \\ \dot{y}(t) = -\sin x(t) \end{cases} \tag{1}$$

portant a terme l'algoritme un nombre `np` de passos. El camp pèndol es té definit en un fitxer apart anomenat `pendol.c`.

Aquest programa imprimeix les posicions a cada instant dels `np` passos. Amb això, podem representar un gràfic de les òrbites per diferents valors inicials i així fer un retrat de fase del camp. Les condicions inicials i nombre de passos utilitzats estan a l'arxiu `rf_pendol.inp`, i els resultats s'han bolcat a l'arxiu `rf_pendol.txt`. El programa s'ha fet córrer amb `hmin = 1e-14`, `hmax = 0.1` i `tol = 1e-15`. El resultat, dibuixat amb `gnuplot` és el que es mostra a la Figura 1.

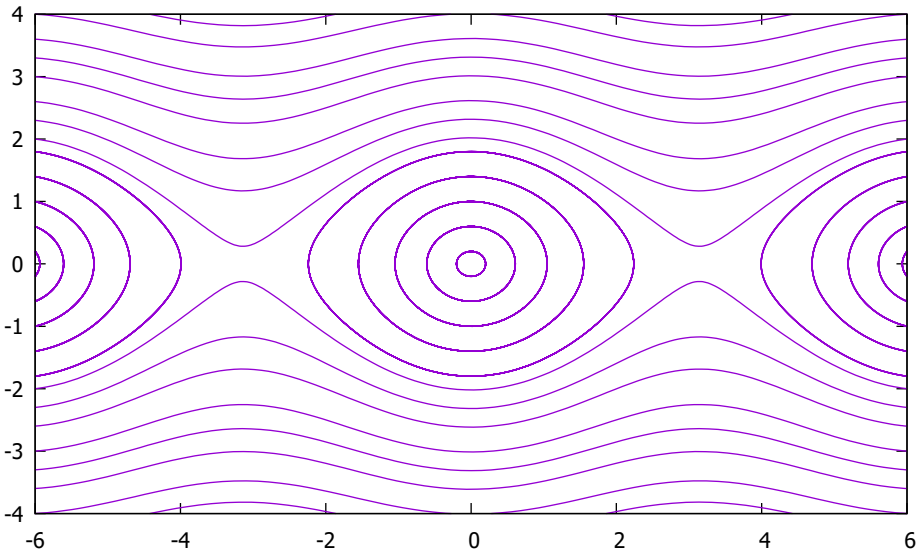

Figura 1: Retrat de fase del pèndol simple

3-4.) El següent pas és fer una rutina `flux()` que troba la solució d'un camp, donades unes condicions inicials, passat un temps `T` donat com a argument. Aquesta funció està implementada a l'arxiu `flux.c`. S'han desenvolupat dos programes curts `flux_check1` i `flux_check2` per comprovar el funcionament d'aquesta rutina amb els problemes de valor inicial:

$$\begin{cases} \dot{z}(t) = \frac{2z(t)}{t} \\ z(1) = 1 \end{cases} \tag{2}$$

$$\begin{cases} \dot{x}(t) = y(t) \\ \dot{y}(t) = -x(t) \\ x(0) = 0, y(0) = 1 \end{cases} \tag{3}$$

les solucions dels quals són conegudes: $z(t) = t^2$ per la primera i $x(t) = \sin t, y(t) = \cos t$ per la segona (oscil·lador harmònic). En aquests programes es mostra l'error comès per `flux()` contra les solucions exactes. Com és d'esperar, l'error és de l'ordre de la tolerància fixada[1] sempre que no es facin ajustos al pas màxim; en aquest cas l'error és molt més petit que la tolerància demanada ja que es prenen passos `hmax` que són més petits del que es necessitaria per assolir aquesta tolerància. Per exemple, amb el primer dels camps si imposem una tolerància `tol = 1e-10` amb pas màxim `hmax = 1` l'error per `T = 3` (que no hagi de fer massa passos) és $-4.3 \cdot 10^{-10}$ -del mateix ordre que la tolerància- mentre que si posem `hmax = 0.1` l'error és $-2.9 \cdot 10^{-12}$ -molt més petit-. Ja es pot veure a la mateixa consola que el `rk78()` ha ajustat al pas màxim en cada pas i és per això que li *sobra* precisió. En conclusió, els programes aquests són una bona prova del bon funcionament de la funció `flux()` que és el que es buscava.

5.) A continuació utilitzem la funció `flux()` per fer una representació de atractor de Lorenz[2]. Considerem les equacions de Lorenz:

$$\begin{cases} \dot{x}(t) = \sigma(y(t) - x(t)) \\ \dot{y}(t) = -x(t)z(t) + \rho x(t) - y(t) \\ \dot{z}(t) = x(t)y(t) - \beta z(t) \end{cases} \tag{4}$$

El programa `lorenz_int.c` calcula punts equiespaiats temporalment entre 0 i un temps donat `tf`. Noti's que el sistema és autònom i per tant és indiferent que es comenci sempre per $t = 0$. S'ha fet córrer el programa seguint les indicacions del guió, amb $\sigma = 3$, $\rho = 26.5$, $\beta = 1$ i punt de partida $x_0 = -0.4164607449115608$, $y_0 = -0.9089362634520914$, $z_0 = 0.01438311162938695$, per obtenir una bona representació de l'atractor de Lorenz. El resultat es mostra representat a la Figura 2.

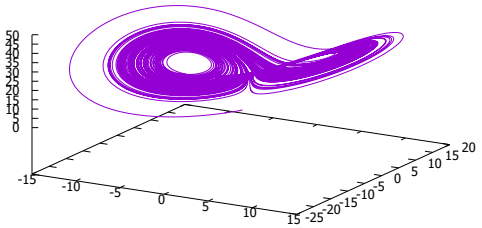

Figura 2: Atractor de Lorenz

[1] No sempre és més petit, ja que la tolerància és fita de l'error per cada pas del `rk78()` i `flux()` duu a terme molts passos del `rk78()`, però si ha de ser més petit que la tolerància multiplicada pel nombre de passos fets.

[2] No s'especifica en cap punt del guió amb quins valors de `h0`, `hmin`, `hmax`, `tol` i `npmax` s'ha de treballar. Per aquest programa i en endavant, sempre que utilitzem la funció `flux()` ho farem amb `h0 = 1e-8`, `hmin = 1e-14`, `hmax = 1`, `tol = 1e-15` i `npmax = 1000`.

6.) Per acabar amb aquesta secció, fem un programa per integrar el camp del problema que ens ocupa. El programa `rtbps_int.c` és un programa anàleg a `lorenz_int.c`, preparat per dibuixar òrbites del problema restringit de 3 cossos. Les equacions que defineixen aquest camp, en coordenades sinòdiques, són

$$\begin{cases} \dot{\mathbf{r}} = \mathbf{v} \\ \dot{\mathbf{v}} = \mathbf{a}(\mathbf{r}, \mathbf{v}) \end{cases} \tag{5}$$

on, si $\mathbf{r} = (x, y, z)^t$ i $\mathbf{v} = (u, v, w)^t$ i donat el paràmetre μ, s'escriu:

$$\begin{aligned} \mathbf{a}(\mathbf{r}, \mathbf{v}) &= 2(v, -u, 0)^t + \nabla\Omega(\mathbf{r}) \\ \Omega(\mathbf{r}) &= \frac{x^2 + y^2}{2} + \frac{1-\mu}{\rho_1(\mathbf{r})} + \frac{\mu}{\rho_2(\mathbf{r})} + \frac{1}{2}\mu(1-\mu) \\ \rho_1(\mathbf{r}) &= \left((x-\mu)^2 + y^2 + z^2\right)^{1/2} \\ \rho_2(\mathbf{r}) &= \left((x-\mu+1))^2 + y^2 + z^2\right)^{1/2} \end{aligned}$$

Aquest camp es té definit a l'arxiu `rtbps.c` que incloem a `rtbps_int.c`. Un aspecte molt interessant d'aquest sistema és la presència de 5 punts d'equilibri, anomenats *punts lagrangians*. Al voltant d'aquests, es troben òrbites periòdiques, anomenades *halos*, que una sonda podria prendre per mantenir-se enmig dels dos cossos.

Amb el programa `rtbps_int.c`, passant-li com a input l'arxiu `halos.inp`, s'han dibuixat diversos exemples d'aquests *halos* en el sistema Terra-Lluna. A la Figura 3 es mostren els resultats: a l'esquerra, una única òrbita on es pot veure la posició de la Terra, la Lluna i el punt L1 respecte aquesta; a la dreta, el conjunt de totes les òrbites de l'arxiu `halos.inp` de forma ampliada.

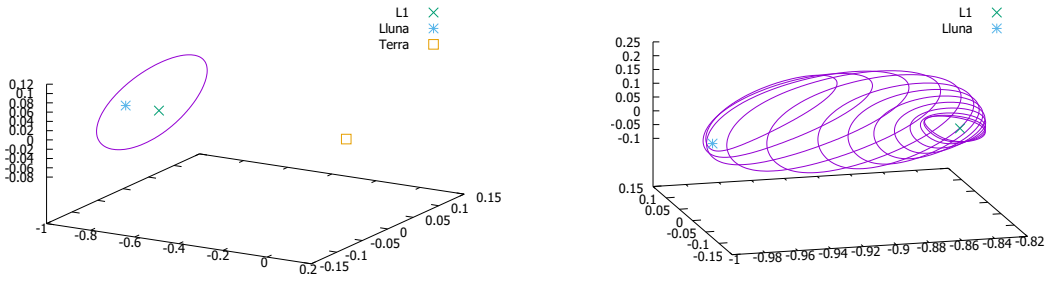

Figura 3: *Halos* del sistema Terra-Lluna.

2 Les variacionals primeres i la diferencial del flux

A la secció 4 necessitarem avaluar la diferencial del flux respecte condicions inicials. En aquesta secció comprovarem que aquestes derivades es poden calcular com a solucions d'un sistema ampliat del camp amb les variacionals primeres d'aquest.

1.) Considerem el problema de valor incial[3] de $n = 2$

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = f(x, y) := \begin{pmatrix} \alpha\left(1 - r^2\right) x - y \\ x + \alpha\left(1 - r^2\right) y \end{pmatrix} \tag{6}$$

[3]Per alleugerir notació, denotarem senzillament x, y en comptes de $x(t), y(t)$.

on $r^2 = x^2 + y^2$. Suposem que les condicions inicials del sistema anterior les escrivim en la forma general $x(t_0) = x_0$ i $y(t_0) = y_0$.

Descrit amb aquesta notació vectorial, les equacions de les variacionals primeres s'escriuen a partir de la matriu diferencial de la funció f. El nou sistema tindrà $n \cdot n = 4$ noves incògnites $a_{00}, a_{01}, a_{10}, a_{11}$ que, escrites en forma de matriu $A = (a_{ij})_{0 \leq i,j \leq 1}$, responen a l'equació donada per $\dot{A} = Df(x, y)A$ amb valors inicials $A(t_0) = I_2$. En resum, ens queda un sistema de $n + n \cdot n = 6$ tal com:

$$\begin{cases} \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} \alpha\left(1 - r^2\right) x - y \\ x + \alpha\left(1 - r^2\right) y \end{pmatrix} \\ \\ \begin{pmatrix} \dot{a_{00}} & \dot{a_{01}} \\ \dot{a_{10}} & \dot{a_{11}} \end{pmatrix} = \begin{pmatrix} \alpha(1 - r^2) - 2\alpha x & -2\alpha xy - 1 \\ -2\alpha xy + 1 & \alpha(1 - r^2) - 2\alpha y \end{pmatrix} \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \\ \\ x(t_0) = x_0,\ y(t_0) = y_0, \\ A(t_0) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{cases} \tag{7}$$

El que es vol demostrar en aquesta part és que, donats t_0, x_0, y_0, les quatre últimes components de la solució del PVI de (7) es corresponen amb les derivades parcials del flux de (6) respecte les condicions inicials als valors x_0 i y_0 donats. Formalment, el que es proposa és que, si $\mathbf{z}(t) = (z_1(t), z_2(t), z_3(t), z_4(t), z_5(t), z_6(t))^t$ és la solució del PVI de (7) i $\phi(t; t_0, x_0, y_0)$ el flux de (6), llavors

$$D\phi(t; t_0, x_0, y_0) = \begin{pmatrix} \frac{\partial \phi_x}{\partial x_0}(t; t_0, x_0, y_0) & \frac{\partial \phi_y}{\partial x_0}(t; t_0, x_0, y_0) \\ \frac{\partial \phi_x}{\partial y_0}(t; t_0, x_0, y_0) & \frac{\partial \phi_y}{\partial y_0}(t; t_0, x_0, y_0) \end{pmatrix} = \begin{pmatrix} z_3(t) & z_5(t) \\ z_4(t) & z_6(t) \end{pmatrix}$$

Per fer-ho, s'ha desenvolupat un programa `variacionals.c` que conté els camps (6) i (7) -accedeix a un o a un altre en funció de si el paràmetre `n` és 2 (accedeix a (6)) o 6 (accedeix a (7))-. Aquest programa utilitza $\alpha = 0.5$, $t_0 = 0$, $x_0 = 1$ i $y_0 = 0$ i, donat un $\delta > 0$, d'una banda deriva numèricament (amb diferències finites centrades de primer ordre en δ) el flux de (6) a $t = 0.5$ respecte x_0 i y_0 i, d'altra banda, calcula aquestes derivades com a solucions del sistema (7) a $t = 0.5$. La sortida del programa és una línia amb els errors en les quatre components de la diferencial del flux. Amb aquest programa hom pot veure que, efectivament, les diferències tendeixen a 0 quan prenem δ prou petita.

A més, pot veure's que aquests errors disminueixen quadràticament amb δ (si disminuïm δ de 0.1 a 0.01 l'error es redueix en totes les components en factor 100, aproximadament). Sabem que el mètode de diferències finites centrades té un error $O(\delta^2)$ amb la solució exacta, de forma que el fet que aquests errors siguin quadràtics amb la solució del sistema (7) ens diu que aquesta segona és una molt bona estimació de les derivades del flux.

```
                delta = 1
err = 0.062038 0.033891 0.073289 -0.134154
               delta = 0.1
err = 0.001009 0.000551 0.000940 -0.001721
              delta = 0.01
err = 0.000010 0.000006 0.000009 -0.000017
             delta = 0.001
err = 0.000000 0.000000 0.000000 -0.000000
```

Figura 4: Sortida del programa `variacionals.exe` per diversos valors de δ.

3 Mètode QR pera sistemes sobredeterminats

A l'hora de resoldre el problema, cal implementar el mètode de Newton (ho tractarem amb detall al següent apartat) en el qual, com ja sabem, cal resoldre un sistema d'equacions lineals. En aquesta secció es desenvolupen funcions que implemente el mètode QR per la resolució de sistemes (determinats o sobredeterminats) mitjançant matrius de Householder.

1.) El primer que es fa és la funció `qrres()` (a l'arxiu `qrres.c`) que és una transcripció literal de l'algoritme de la pàgina 14 del guió. Donada una matriu $A \in M_{m\times n}(\mathbb{R})$ amb $m > n$ i un vector $b \in \mathbb{R}^m$, el procediment `qrres()` troba la solució (de mínims quadrats, si es sobredeterminat) al sistema $Ax = b$ i ens retorna també, de forma indirecta entre els valors de sortida, la matriu R i els vectors $u_0, ..., u_{n-1}$ a partir dels quals fer les reflexions de Householder que donen la descomposició $A = QR$.

2.) A continuació es comprova el funcionament de la funció `qrres()` amb

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

S'ha desenvolupat un petit programa anomenat `qresexemple.c` que aplica la funció `qrres()` sobre les matrius que li entrem. Quan entrem les matrius A i b anteriors, el que obtenim a la sortida de `*a`, `*b`, `*dr` i `*x` és, respectivament:

$$\texttt{*a} = \begin{pmatrix} 5 & 2 \\ 0 & 4 \\ 5 & 4 \end{pmatrix} \qquad \texttt{*b} = \begin{pmatrix} -2 \\ 1 \\ -3 \end{pmatrix} \qquad \texttt{*dr} = \begin{pmatrix} -5 \\ -4 \end{pmatrix} \qquad \texttt{*x} = \begin{pmatrix} 0.3 \\ -0.25 \end{pmatrix}$$

on es pot veure que els resultats concorden amb el que ha de ser: a `*dr` tenim la diagonal de R, a la part de sobre de la diagonal de `*a` (només el 2) tenim la part de sobre de la diagonal de R, a la part de la diagonal i inferior de `*a` els vectors u_k de cada reflexió posats en columna, a `*b` el terme independent després de les transformacions, $b^{(2)}$, i a `*x` la solució al sistema (en aquest cas no sobredeterminat ja que, tot i que A té més files que columnes, el seu rang és igual a n). Tant $R = A^{(2)}$, com $b^{(2)}$, com x són els que esperàvem (donats al guió), de forma que es veu com `qrres()` funciona correctament.

3.) En aquest apartat avaluem la precisió de l'algoritme enfront matrius de dimensions grans ($n \geq 100$). Per això, s'ha generat un programa `qrerror.c` que, donat n, genera una matriu A de dimensió $n \times n$ amb elements aleatoris amb distribució uniforme entre 0 i 1, i resol -mitjançant `qrres()`- un sistema $Ax = b$ que tingui com a solució $x = (1, ..., 1)^t$. Per fer això, només cal prendre $b = \left(\sum_{j=1}^{n} a_{ij}\right)_{i=1,...,n}$, i.e., el vector de termes independents serà les sumes per files de la matriu A.

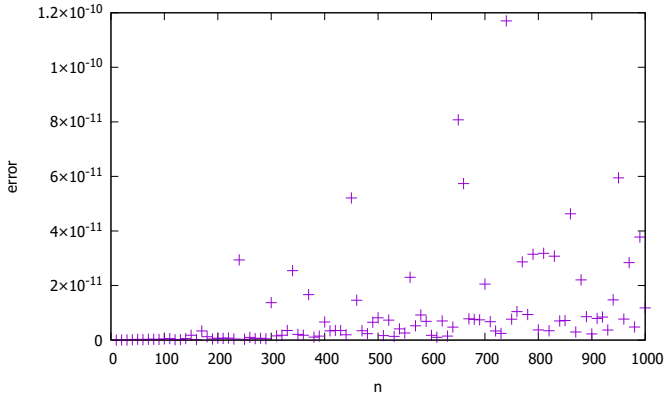

Figura 5: Error del mètode QR per matrius aleatòries en funció de la dimensió n.

Els errors obtinguts pel programa, evidentment, són aleatoris ja que depenen de la matriu aleatòria A. Podríem córrer el programa dos cops amb el mateix valor de n i el resultat que obtindríem seria diferent. S'ha fet córrer el programa amb valors de n desde 10 fins a 1000 de 10 en 10 (hem generat de forma automàtica el fitxer `qrerror.inp`) i a la Figura 5 es mostren els resultats obtinguts. No es té un creixement estricte de l'error enfront n, ja que l'error no ve determinat per n però sí podem veure com, en general, l'error tendeix a créixer quan n creix. Una recta d'ajust per mínims quadrats ens dóna un pendent $(2.39 \pm 0.58) \cdot 10^{-14}$, que ens diu que aquest pendent (aquesta relació entre el creixement de l'error amb n) és positiu -de fet està entre 1.81 i 2.97 amb una confiança estàndard (68.27%)-.

Tenim evidències suficients per dir que l'error augmenta amb n, la qual cosa és natural degut a que, en augmentar n, augmenta el nombre d'operacions que es fan, les qual tenen un error associat que va acumulant-se. Tot i així, val a dir que aquests errors no superen mai un ordre de 10^{-9} i la seva mitja ens els casos avaluats està al voltant de 10^{-12}.

4.) Per acabar amb aquesta part, parlarem l'eficiència computacional de l'algoritme. El programa `qrtemps.c` és una petita variació de `qrerror.c` que calcula el temps emprat en fer l'algoritme QR (temps emprat per la funció `qrres()` amb `b=NULL`) per una matriu quadrada A de mida n donada. Passant-li el mateix imput que a l'apartat anterior, obtenim un resultat com el que mostra la Figura 6.

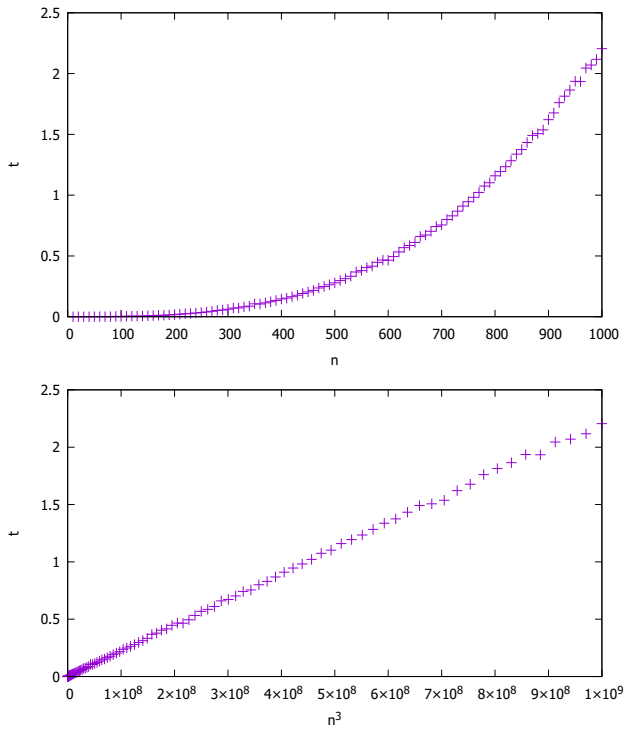

Figura 6: Temps d'execució de l'algoritme QR enfront n i n^3.

És ben clar que aquest temps d'execució no creix pas de forma lineal amb n. Concretament, tenim que el nombre d'operacions de l'algoritme QR és $\frac{4}{3}n^3 + O(n^2)$. A la Figura 6 es mostra també una representació del temps enfront n^3 que posa en evidència aquesta dependència cúbica.

El programa `qrtemps.c` també calcula el rendiment efectiu del programa, considerant que el nombre d'operacions és $\frac{4}{3}n^3$ (aproximació prou bona per n prou gran). Un simple quocient entre el nombre d'operacions $\frac{4}{3}n^3$ i el temps cronometrat t ens diu quin ha estat la freqüència efectiva a la que ha treballat el programa. Dividint entre la freqüència teòrica de l'ordinador (2.6 GHz) n'obtenim una estimació del rendiment. Per un n prou gran ($n > 100$) el resultat d'aquest rendiment està sempre al voltant de 0.23, i.e., un 23%.

4 Càlcul de les maniobres

Finalment, en aquesta última secció fem ús de tots els codis que s'han anat desenvolupant per resoldre el problema que ens ocupa: donades unes condicions inicials $\mathbf{r}_0, \mathbf{v}_0 \in \mathbb{R}^3$, un interval de temps $\Delta t \in \mathbb{R}$ i unes condicions finals $\mathbf{r}_f, \mathbf{v}_f \in \mathbb{R}^3$, quines maniobres $\Delta\mathbf{v}_0, \Delta\mathbf{v}_1 \in \mathbb{R}^3$ (efectuades a $t = 0$ i $t = \Delta t/2$, respectivament) portaran un objecte, sotmès al problema restringit de 3 cossos (d'un cert paràmetre μ), des de $\mathbf{r}_0, \mathbf{v}_0$ fins a $\mathbf{r}_f, \mathbf{v}_f$ en un temps Δt?

1-2.) Podem dividir el trajecte en 4 etapes: un primer impuls donat per $\Delta\mathbf{v}_0$, vol lliure des de $t = 0$ fins a $t = \Delta t/2$, el segon impuls donat per $\Delta\mathbf{v}_1$ i per últim vol lliure des de $t = \Delta t/2$ fins a $t = \Delta t$. Si denotem per $\phi_t(x) := \phi(t; 0, x)$ el flux del camp RTBP passat un temps t amb condicions inicials x (noti's que és independent fixar $t = 0$ per les condicions inicials ja que el camp és autònom), les quatre etapes les podem escriure com:

$$\begin{pmatrix} \mathbf{r}_0 \\ \mathbf{v}_0 \end{pmatrix} \to \begin{pmatrix} \mathbf{r}_0 \\ \mathbf{v}_0 + \Delta\mathbf{v}_0 \end{pmatrix} \to \phi_{\Delta t/2} \begin{pmatrix} \mathbf{r}_0 \\ \mathbf{v}_0 + \Delta\mathbf{v}_0 \end{pmatrix} \to$$
$$\phi_{\Delta t/2} \begin{pmatrix} \mathbf{r}_0 \\ \mathbf{v}_0 + \Delta\mathbf{v}_0 \end{pmatrix} + \begin{pmatrix} 0 \\ \Delta\mathbf{v}_1 \end{pmatrix} \to \phi_{\Delta t/2} \left(\phi_{\Delta t/2} \begin{pmatrix} \mathbf{r}_0 \\ \mathbf{v}_0 + \Delta\mathbf{v}_0 \end{pmatrix} + \begin{pmatrix} 0 \\ \Delta\mathbf{v}_1 \end{pmatrix} \right)$$

I el terme final serà la posició final de la nau en funció dels impulsos $\Delta\mathbf{v}_0, \Delta\mathbf{v}_1$. Trobar quins són aquests $\Delta\mathbf{v}_0, \Delta\mathbf{v}_1$ tals que aquest últim terme sigui $(\mathbf{r}_f, \mathbf{v}_f)^t$ equival a trobar el 0 de la funció

$$\mathbf{G}(\Delta\mathbf{v}_0, \Delta\mathbf{v}_1) = \phi_{\Delta t/2} \left(\phi_{\Delta t/2} \begin{pmatrix} \mathbf{r}_0 \\ \mathbf{v}_0 + \Delta\mathbf{v}_0 \end{pmatrix} + \begin{pmatrix} 0 \\ \Delta\mathbf{v}_1 \end{pmatrix} \right) - \begin{pmatrix} \mathbf{r}_f \\ \mathbf{v}_f \end{pmatrix}$$

i ho farem pel mètode de Newton.

En els apartats 1 i 2 d'aquesta secció es realitzen dues rutines `cmani_gdg()` i `cmani()` (a l'arxiu `cmani.c`): la primera calcula, donats $\Delta\mathbf{v}_0, \Delta\mathbf{v}_1$, el valor de la funció $\mathbf{G}(\Delta\mathbf{v}_0, \Delta\mathbf{v}_1)$ i de la seva diferencial $D\mathbf{G}(\Delta\mathbf{v}_0, \Delta\mathbf{v}_1)$; la segona fa ús d'aquesta primera per implementar el mètode de Newton, amb una llavor, tolerància i nombre màxim d'iteracions donats com a arguments, i trobar les $\Delta\mathbf{v}_0, \Delta\mathbf{v}_1$ que anul·len $\mathbf{G}(\Delta\mathbf{v}_0, \Delta\mathbf{v}_1)$ i són per tant solució del problema.

Aquestes dues funcions fan ús de la funció `flux()` que hem desenvolupat al primer apartat, calculen la diferencial de $\mathbf{G}$ a partir de variacionals primeres com hem vist a la secció 2, i empren el mètode QR de la secció 3 quan aplica el mètode de Newton per a trobar $D\mathbf{G}(\Delta\mathbf{v}_0, \Delta\mathbf{v}_1)^{-1}\mathbf{G}(\Delta\mathbf{v}_0, \Delta\mathbf{v}_1)$ com a solució del sistema $D\mathbf{G}(\Delta\mathbf{v}_0, \Delta\mathbf{v}_1)x = \mathbf{G}(\Delta\mathbf{v}_0, \Delta\mathbf{v}_1)$.

3.) Es posen a prova les funcions primer amb un camp senzill com és el del pèndol simple. Per a poder usar aquest camp en la rutina `cmani()` primer hem de modificar-lo per tal que doni el camp ampliat amb les variacionals primeres quan li passin un argument `n` més gran que 2 (dimensió del sistema del pèndol). El programa `cmani_pendol.c` executa la funció `cmani()` per al camp del pèndol passant $r_0 = 1$, $v_0 = 0$,$r_f = 0$,$v_f = -\sqrt{2(1-\cos 1)}$ amb $\Delta t = \pi/2$. La sortida del programa per consola és la que es mostra a continuació.

```
cmani() ::  it 1 ng 9.98646E-002 nc 9.34007E-002
cmani() ::  it 2 ng 1.54523E-003 nc 1.84063E-003
cmani() ::  it 3 ng 1.93848E-008 nc 2.51935E-008
cmani() ::  it 4 ng 6.12715E-016 nc 1.00213E-015
cmani() ::  dv1 = -0.092698
cmani() ::  dv2 = 0.006208
```

Pot veure's la convergència quadràtica del mètode en el fet que la norma de $\mathbf{G}$ (segona columna) va disminuint elevant-se al quadrat -aproximadament- a cada pas.

4.) Per concloure, apliquem aquestes funcions al camp RTBP per tenir solucions al problema. S'ha creat la utilitat `cmani_rtpb.c` que rep com a arguments la μ (`mu`) del sistema, la tolerància `tol` i el nombre màxim d'iteracions `maxit` per Newton i, per cada línia de la forma `dt x0 y0 z0 v0x v0y v0z xf yf zf vfx vfy`

`vfz` amb el temps i les condicions inicials i finals retorna `dv0x dv0y dv0z dv1x dv1y dv1z` amb els dos impulsos que resolen el problema.

Pot comprovar-se que en bolcar les dades de `cmani_rtbp.inp` facilitades amb `mu = 1.2150585609624e-2`, `tol = 1e-12` i `maxit = 10`, el resultat obtingut és:

```
-8.32618E-006 -1.28566E-005 -1.54588E-006 -5.98913E-006 -1.07116E-005 -1.80927E-006
```

que coincideix novament de forma exacta amb l'arxiu `cmani_rtbp.out` facilitat.

Per últim, hem volgut fer un dibuix de la trajectòria que dóna la solució, amb els punts d'inici i d'arribada i la posició de la Terra i la Lluna. El resultat es mostra a la Figura 7.

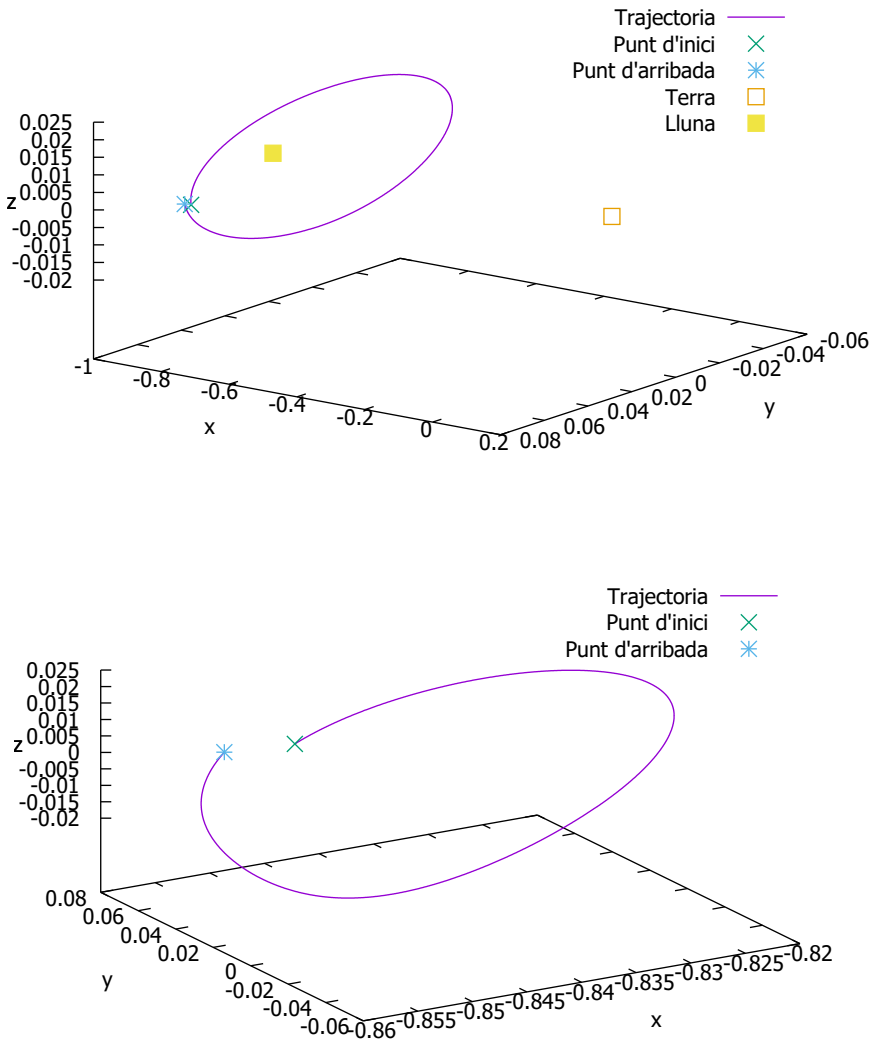

Figura 7: Trajectòria de la solució en el sistema Terra i Lluna.