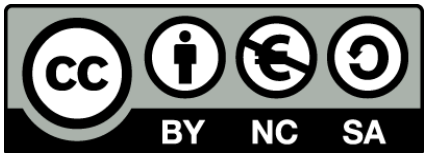
UNIVERSITAT DE BARCELONA

Dynamical transport mechanisms in celestial mechanics and astrodynamics problems

Daniel Pérez Palau



Dynamical transport mechanisms in celestial mechanics and astrodynamics problems

Daniel Pérez Palau

PhD Advisors: **Gerard Gómez Muntané**
and **Josep J. Masdemont Soler**

Universitat de Barcelona

Programa de doctorat en Matemàtiques

Tesi presentada per en Daniel Pérez Palau per optar al grau de
Doctor en Matemàtiques

Directors: Gerard Gómez Muntané i Josep J. Masdemont Soler

Departament de Matemàtica Aplicada i Anàlisi
Facultat de Matemàtiques
Setembre de 2015

Certifico que la següent tesi
ha estat realitzada per en
Daniel Pérez Palau
sota la meva codirecció.

Barcelona, setembre de 2015.

Gerard Gómez Muntané

Certifico que la següent tesi
ha estat realitzada per en
Daniel Pérez Palau
sota la meva codirecció.

Barcelona, setembre de 2015.

Josep J. Masdemont Soler

A Irene

Agraïments

Es pot comparar la superació d'una tesi doctoral a creuar un gran riu. L'objectiu és arribar a l'altre riba i per fer-ho només disposem de pedres, moltes pedres que sobresurten del riu i ens permeten anar fent passes de mica en mica. A cada nou pas que vols fer cal temptejar el terreny, mirar si la pedra a la que vols passar és segura. Relliscarà? Serà ferma? A vegades és possible afegir petits rocs per a falcar-la i aconseguir un millor pas. D'altres només queda la opció de fer un salt de fe i confiar en l'experiència, la intuïció i la sort per a que tot surti bé.

Per sort creuar, aquest riu no s'ha de creuar en solitari. Hi ha molta gent disposada a ajudar-te. És per això que m'agradaria dedicar unes línies a tots aquells companys, amics i familiars que m'han donat suport, ja sigui a nivell acadèmic com personal.

Les primeres d'aquestes línies les vull dedicar als meus directors de tesi: en Gerard i en Josep. Ells han sigut un suport cabdal en la realització d'aquesta tesi. M'han ajudat a salvar les dificultats i de la travessa i a triar quines pedres són les adients per a continuar. Sempre han estat disposats a donar-me un cop de mà, i sobretot, durant les reunions dels dimecres. Haig de reconèixer que les reunions dels dimecres em feien una certa por de bon principi. Al cap de poc temps vaig adonar-me de quan interessants podien resultar. La veritat és que trobaré a faltar aquesta reunió setmanal i sobretot la trencadissa de cap per trobar una bona imatge i un bon títol per a introduir la presentació de la setmana. Encara ara, després de més de 130 trobades i incomptables fotografies i títols arribo a la reunió pensant si agradarà o no la imatge que he triat per a la setmana, però de seguida comencem les "discussions" i explicacions sobre la feina feta. És en aquest moment quan m'han donat inspiració per provar noves vies d'atac, noves pedres a les quals saltar i en definitiva ajudar-me a acabar creuant el riu.

També voldria agrair l'ajuda i l'acolliment rebut per part dels membres del departament: l'Àngel i en Carles per desencallar i mostrar-me la pedra sobre la que calia saltar a continuació; a l'Ernest per totes les facilitats que m'ha donat a l'hora de tramitar papers i burocràcia; a l'Arturo per la infinitud de discussions científiques i preguntes que m'han ajudat a comprendre quins objectius perseguia; i a en Jaume per tot el suport informàtic que m'ha estalviat hores i hores resolent gairebé al moment els problemes que algun ordinador m'ocasionava. També vull donar les gràcies a la Montse per ajudar-me en les primeres classes que em va tocar fer i a l'Àlex per fer-me gaudir de les darreres. Dentro del departamento, también merece una mención especial Ino, nuestra secretaria por ayudar a organizar viajes y burocracia con la universidad.

Passant a un punt més personal, però sense deixar el departament ni la universitat no em vull deixar els companys de despatx, en David i en Marc, així com en Fabrizio i en Rubén. Junts hem compartit moltes hores de les nostres vides així com grans debats a estones mortes i petites i enriquidores discussions acadèmiques per trencar la monotonia de la tarda. També els hi haig d'agrair el suport moral en moments de desesperació que la tesis ha portat.

També em vull recordar en aquestes línies de tot el grup de doctorands, postdocs i becaris de tota mena amb qui he conviscut i passat molt bones estones durant aquests darrers anys: l'Andratx, en Carlos, l'Eloi, en Dani, la Giulia, en Lian, la Marta, la Meri, la Nadia, la Neus, la Priscila, en Roc, la Rocío, la Stefania, la Ting Ting, en Tomasso i la Yu. Amb tots ells he compartit dinars, cafès i begudes després d'una jornada intensa. Fins i tot, vàrem arribar a mantenir un seminari, a part de la seva vessant acadèmica també ens va ajudar a cohesionar el grup. Segur que sense vosaltres no hagués estat possible. Però en especial vull agrair a: en Narcís tots els debats i intents que hem fet d'iniciar noves activitats de recerca així com per les tardes senceres dedicades a donar un cop de mà en els moments més necessitats tot comentant la jugada

sota l'ombra d'un arbre o amb unes bones braves; a l'Ari per la guia i el referent en aquests primers anys de recerca i per mostrar-me que sempre cal tenir una actitud positiva davant la vida; a en Jordi per les estones de geocaching i d'evasió general, que tot i que sembla que no, són tan necessàries; and finally to Zubin with who I had a lot of interesting discussions and debates as well as for his patience with my stupid English questions.

In the international panorama I would like to thanks to the Multi-Body Dynamics Research Group at Purdue University with who I spent a couple of weeks in summer 2012, specially to professor K. Howell and Cody for their warm reception.

Also specials thanks to the Astronet-II group, although I was not a proper member of astronet they make me feel like one of them: Albert, Alex, Andrea, Claudiu, Elisabetta, Luca, Marta and Willem.

Tornant a Barcelona i anant uns quants anys enrere també vull agrair als companys de la carrera: en David R., la Kaothar, la Marta Castellano, la Roser i en Xavier. Amb ells vàrem començar a saltar les primeres pedres que em portarien a acabar creuant aquest riu. Sense ells la carrera haguera estat més dura.

També en èpoques passades vull agrair als meus professors tan de la carrera, la Núria i l'Antòn, que em van motivar a fer el doctorat, com d'etapes anteriors per animar-me a començar la carrera. En especial, a en Llorenç Rosselló per treure'ns del pecat i portar-nos al camí del bé, amb sort, el projecte que ara acaba m'haurà redimit d'aquelles divisions per zero de la secundària

Però creuar el riu no està exempt de perill. No són pocs els dies en que t'arrisques, o potser no tant, en fer un salt i surt malament. Aleshores vas a parar de pet a l'aigua i és en aquest moment en el que busques el suport dels teus. De la gent que és a casa. Són dies negres en els que tot el que tens ganes és d'enviar-ho tot a pastar fang, dir bon vent i barca nova. Però saps que sempre te'ls trobes allà per animar-te. Quiero agradecer a mi padre por toda la motivación y el pensamiento crítico que me acabaron llevando a empezar y acabar la carrera y después continuar con el camino del doctorado así como por "cómo ha ido hoy?" diario al que demasiadas veces he respondido de malas maneras. A la mare i en Jordi, per saber-me escoltar i fer costat en moments de crisi. Als meus germans: la Marina, per les nostres converses a les dues de la matinada que fins i tot volia repetir mentre escribia aquestes línies tot i estar a mes de mil quilometres de casa; i en Pol per aportar la teva rialla innocent i deixar-me fer de germà gran. També vull agrair als avis, a qui no vaig a veure tan com m'agradaria, per les seves visites sorpresa. A la *tata* por las horchatas en la valenciana y los suministros no solo navideños que han hecho la logística del día a día más fácil. Així com tots els tiets, tietes, cosins i cosines per les trobades, sortides, dinars, vacances, calçotades i sopars en família que hem passat junts tots aquest anys i han aportat petites estones d'evasió a la rutina diària. Ahora que me acerco al final de estas lineas no puedo evitar acordarme de los hace tiempo que ya no están, y que seguro que les hubiera hecho ilusión estar.

Finalmente, quiero agradecer a Irene el apoyo, la confianza y los ánimos que me ha dado durante estos cuatro años y especialmente la paciencia de este último año con mis constantes cambios de humor y de ánimo. También quiero agradecerle los buenos momentos y escapadas que me han resultado más que útiles y a cuya vuelta no era extraño encontrar la solución a problemas que llevaban estancados más tiempo del deseado. Y aunque parece contradictorio esos fines de semana alejado del trabajo acababan reportando más beneficios que una semana entera de trabajo constante.

Crec que no m'he deixat a ningú pel camí. Sé que es podrien fer recol·locacions i moviments però m'ha quedat així, uns agraïments és quelcom que ha de sortir de dins i així és com ha sortit. Sé que tot lector que arribi fins a aquestes línies sap prou de mi com per recol·locar-se ell mateix en els agraïments que li corresponen.

Moltes gràcies a tots!

Dani
Barcelona, 30 de setembre de 2015

Esta tesis ha sido posible gracias a la beca de Formación del Profesorado Universitario AP2010-0268 y los proyectos MTM-2010-16425 y MTM2013-41168-P otorgados por el Ministerio de Educación Cultura y Deporte del Gobierno de España

Resum en Català

L'objectiu d'aquesta tesi és afegir un petita fulla a l'abre del coneixement. En particular a la branca del sistemes dinàmics.

La teoria de sistemes dinàmics és la branca de les matemàtiques que estudia l'evolució del que ens envolta. Un dels objectius de la teoria dels sistemes dinàmics és estudiar com evoluciona amb el temps un cert procés evolutiu, és a dir, donades unes condicions inicials per a un cert estat, quin serà l'estat del sistema t unitats de temps. En alguns problemes és possible trobar estructures que ens separen diferents tipus de moviment. Per exemple, un moviment fitat d'un de no fitat. Aleshores, aquestes estructures determinen com evoluciona el sistema sota estudi. És en aquest cas parlem de mecanismes dinàmics de transport. És a dir, quines són les possibles maneres que té un cert estat d'arribar a un altre.

La teoria de sistemes dinàmics treu models i problemes gran varietat d'àmbits científics. En aquesta tesi ens centrarem en problemes de mecànica celeste i astrodinàmica. Ambdues disciplines es centren en l'estudi d'objectes a l'espai. La mecànica celeste es basa en les lleis del moviment i de la gravitació universal de Newton. A partir d'aquestes lleis bàsiques és possible estudiar com es mouen els cossos a l'espai. Per exemple un planeta al voltant de la seva estrella. En canvi, si el que es vol estudiar és el moviment d'una nau espacial, les seves trajectòries i quines maniobres haurà de fer per arribar a un cert lloc, aleshores parlem d'astrodinàmica o mecànica orbital.

En ambdós casos, els sistemes que estudiarem són sistemes dinàmics continus, descrit mitjançant una equació diferencial definida pel camp vectorial f,

$$\dot{\vec{x}} = f(t, \vec{x}),$$

que en cada posició $\vec{x}$ i instant de temps t ens dona quina és la variació instantània d'un objecte en aquesta posició i temps.

Anomenem *flux* a la solució d'aquesta equació i la designem per $\phi(t; t_0, x_0)$ que ens dona la posició a temps t d'una partícula que en temps t_0 es troba en x_0. Sovint no és possible trobar la funció ϕ de forma analítica. Quan això succeeix no tot està perdut. Encara podem calcular una bona aproximació de la solució mitjançant tècniques de resolució d'equacions diferencials numèriques. La tècnica més simple és coneguda amb el nom de *mètode d'Euler* i consisteix en usar el camp vectorial i petits increments de temps per tal de conèixer la situació de l'objecte estudiat en un temps futur. També és possible usar mètodes més complexos per aproximar la solució. En aquest cas ens cal trobar la serie de Taylor de la solució en funció del temps a partir del camp vectorial i les seves derivades. Alguns d'aquests mètodes usen fórmules tancades per trobar l'aproximació a un ordre desitjat. En són un exemple els mètodes de Runge-Kuta (RK). Malauradament, si es vol obtenir una aproximació d'ordre més alt cal calcular de nou les fórmules.

Per evitar aquest problema es poden usar mètodes de Taylor. Aquests mètodes calculen els diferents ordres de l'aproximació de la solució mitjançant diferenciació automàtica. D'aquesta manera és possible evitar el càlcul de fórmules tancades per a trobar aproximacions de les solucions.

Així doncs, mitjançant la resolució numèrica podem predir quina serà la situació d'una certa partícula al cap d'un cert temps. En algunes situacions també ens interessarà saber, no només que li succeeix a la nostra partícula sinó que els hi succeeix a les partícules que l'envolten. Aquest coneixement es pot extreure de la solució de les equacions variacionals.

Les equacions variacionals es poden obtenir derivant el camp vectorial. A primer ordre venen donades

per

$$\begin{cases} \dot{J}(t) = D_x f(t,x) J(t), \\ J(0) = Id, \end{cases}$$

on $J = D_{\vec{x}_0}\phi(t;t_0,\vec{x}_0)$. Un cop determinada al solució, $J(t)$ podem saber la situació futura de punts propers a la partícula mitjançant

$$\phi(t;t_0,x_0+\vec{\xi}) = \phi(t;t_0,\vec{x}_0) + D_{\vec{x}_0}\phi(t;t_0,\vec{x}_0)\vec{\xi}.$$

Si volem obtenir solucions a ordre més alt, haurem de calcular successives derivades del camp vectorial fins a arribar a l'ordre desitjat.

Tot i això, la determinació de les equacions variacionals a ordres alts resulta feixuga i laboriosa fet que fa que sovint apareguin errors de càlcul. En el Capítol 2 s'introdueix una tècnica alternativa al càlcul d'equacions variacionals que evita la feina a fer, el *Jet Transport*. La idea fonamental del Jet Transport és expandir les solucions mitjançant series de Taylor usant tècniques de diferenciació automàtica, de la mateixa manera que es fa en els mètodes de Taylor per a la resolució d'equacions diferencials. D'aquesta manera s'aconsegueix l'expansió de la solució en les variables espacials en forma de polinomi:

$$P(\vec{\xi}) = \sum a_{(k_1,\ldots,k_l)} \xi_1^{k_1} \ldots \xi_l^{k_l}$$

Indicadors de la dinàmica

Un cop sabem com propagar les condicions inicials mitjançant un camp vectorial l'objectiu és extreure'n la màxima informació del dinàmica global associada.

La primera informació que en podem extreure és la localització dels punts fixos, és a dir, aquells punts $\vec{x}$ tals que $f(\vec{x}) = 0$. Són punts estacionaris del sistema i representen aquells estats que no varien en el temps. Aquests punts poden ser estables o inestables en funció del que succeeix amb punts propers a ells. Si tots els punts al voltant d'un punt fix romanen propers al punt fix per tot temps aleshores es diu que el punt és estable. En canvi, si hi ha algun punt proper que se n'allunya aleshores diem que el punt és inestable.

De manera semblant es pot repetir l'estudi per a la detecció d'òrbites periòdiques, i en general tors invariants. Una òrbita periòdica és una solució de l'equació definida pel camp vectorial $\dot{\vec{x}} = f(t,\vec{x})$ tal que existeix un temps T pel qual $\phi(t_0+T;t_0,\vec{x}_0) = \vec{x}_0$ per tota condició inicial $\vec{x}_0$ a l'orbita periòdica i tot temps t_0. De la mateixa manera que amb els punts fixos les òrbites periòdiques poden ser estables o inestables. Les òrbites periòdiques estables són aquelles en que òrbites properes romanen properes mentre que les prop de les òrbites periòdiques inestables hi ha òrbites que se n'allunyen.

Tan de punts fixos com d'òrbites periòdiques poden emanar varietats invariants. Les varietats invariants són solucions particulars del problema que surten o arriben al punt fix o l'òrbita periòdica.

Tots aquests elements plegats, punts fixos, òrbites periòdiques, tors invariants i varietats invariants formen el que s'anomena esquelet de la dinàmica.

La dinàmica dels punts fixos i òrbites periòdiques és regular i fàcil de preveure. En canvi, hi ha altres regions de l'espai de fase que són caòtiques. Aquestes regions es poden reconèixer perquè donats dos punts en aquestes regions, independentment de quan aprop estiguin entre sí, acabaran tenint un comportament molt diferent.

Així doncs, distingir les regions caòtiques de les regulars és important. Per exemple per a saber que succeeix si desviem lleugerament la condició inicial. Descriurà una òrbita similar o completament diferent?

Aquest tipus de preguntes són altament interessants en problemes de mecànica celeste. Un problema clàssic consisteix en saber si el sistema solar és estable o no. És clar que si només considerem un planeta orbitant la seva estrella aquest es mourà seguint òrbites el·líptiques. Ara bé, si afegim un segon planeta i deixem actuar els tres cossos sota la seva acció gravitatòria mútua, seguiran els planetes descrivint òrbites el·líptiques o properes a el·líptiques? o pel contrari es mouran seguint òrbites molt diferents?

Una eina usual en l'estudi de la regularitat/caoticitat de les solucions d'un sistema dinàmic són els *indicadors de la dinàmica*. Un indicador de la dinàmica és un observable del sistema que ens dona informació

sobre el comportament de les solucions. Un exemple molt popular són els exponents de Lyapunov, introduïts l'any 1892 per A. Lyapunov i definits com:

$$\sigma(\vec{x}_0) = \lim_{t\to\infty} \frac{1}{t} \log \|D\phi(t; t_0, \vec{x}_0)\|.$$

Amb el pas del temps han sorgit altres indicadors de la dinàmica per millorar la precisió dels resultats obtinguts, disminuir el temps de còmput o bé donar alternatives amb similars propietats.

Un exemple d'aquests nous indicadors són els exponents de Lyapunov a temps finit (FTLE). Aquests, han estat fets servir per G. Haller per tal d'establir la teoria de les Estructures Lagrangianes Coherents (LCS). Les LCS són estructures que permeten separar diferents regions de l'espai de fase segons el seu comportament dinàmic. Actuen de forma semblant a com ho fan les varietats invariants, tot i que no són equivalents. En el Capítol 3 es parla de com calcular els FTLE i les LCS i s'introdueixen nous indicadors de la dinàmica tot fent servir la informació obtinguda amb el Jet Transport.

Aplicacions del Jet Transport a l'evitació de col·lisions

Des d'un punt de vista astrodinàmic, la determinació de zones estables també és interessant. Quan una agència espacial llança una nau espacial a l'espai necessita saber si la nau seguirà l'òrbita prevista sense fer cap maniobra addicional o si en caldrà fer-ne alguna.

La dinàmica al voltant de la Terra és rica i complexa. Fins i tot per a períodes curts de temps, una nau orbitant prop de la Terra pateix pertorbacions que no són negligibles. A més a més, cada dia hi ha més objectes orbitant aquesta regió de l'espai. Es calcula que hi ha uns 3 600 satèl·lits artificials i un gran nombre d'escombraries espacials de diferent mida. Així doncs, el risc de patir una col·lisió entre dos d'aquests cossos és present.

Cal estudiar aquestes possibles col·lisions per tal de saber si el risc és alt o no. En cas d'alts riscos d'impacte aleshores cal estudiar quan i quines maniobres caldrà fer per tal de mantenir la probabilitat de col·lisió per sota d'uns valors acceptables.

Aquests estudis es poden fer mitjançant dos mètodes. El primer consisteix en usar aproximacions analítiques de la probabilitat de col·lisió usant propagacions lineals de les funcions de densitat. Aquests mètodes tenen com a inconvenient que l'aproximació obtinguda és pobre degut a que no tenim en compte els efectes no lineals presents al problema.

Per tal de tenir en compte els efectes no lineals es fan servir simulacions de Monte Carlo. En aquest cas, cal propagar multitud de condicions inicials properes per tal d'obtenir la probabilitat desitjada. Degut al gran nombre de propagacions que cal fer el temps de còmput és llarg. En el Capítol 4 s'estudia aquest problema introduint una opció intermitja mitjançant el Jet Transport. D'aquesta manera s'escurcen els temps de còmput i es tenen en compte els efectes no lineals.

Estructura de la Tesi

L'estructura de la tesi és com segueix:

- El Capítol 1 està dedicat a introduir alguns del conceptes que es fan servir en els capítols posteriors, així com qüestions de notació i la definició dels sistemes dinàmics que s'empraran.

- En el Capítol 2 s'introdueix l'eina principal de la tesi, el Jet Transport. Per fer-la servir cal implementar una àlgebra de polinomis. El capítol explica com fer aquesta implementació. Les primeres seccions es dediquen a explicar com fer un ús eficient de la memòria i a introduir les operacions bàsiques amb polinomis (el producte per un escalar, la suma, el producte, la divisió de dos polinomis). També s'explica com realitzar altres operacions elementals com l'exponencial, el logaritme, el sinus i el cosinus així com la derivació i la integració de polinomis. A les darreres seccions s'explica com implementar operacions més complexes com la propagació de fluxos (incloent el càlcul d'aplicacions de Poincaré i

altres tècniques per a millorar els resultats obtinguts), el càlcul de la inversa funcional d'un polinomi i la transformació de densitats mitjançant una aplicació.

- El Capítol 3 està dedicat a parlar sobre indicadors dinàmics. Primer es repassen els exponents de Lyapunov a temps finit i les estructures lagrangianes coherents. Fruit d'aquestes reflexions es desenvolupen algorismes per tal de disminuir el temps de còmput. Tot seguit, es donen quatre indicadors de la dinàmica alternatius basats en el Jet Transport: la màxima mida de la caixa inicial, la màxima relació d'expansió, la màxima relació de contracció i la màxima relació d'expansió a l'espai normal. El capítol segueix desenvolupant un algorisme d'extracció d'estructures per tal d'extreure i resumir la informació donada pels indicadors dinàmics. Finalment, es fan servir els indicador dinàmics introduïts per tal de determinar zones d'estabilitat efectiva en el problema restringit de tres cossos.

- En el Capítol 4 s'estudia la col·lisió de satèl·lits artificials. Primerament s'estudien les diferents pertorbacions que afecten al moviment de satèl·lits al voltant de la terra. Es considera un problema de dos cossos amb pertorbacions degudes al potencial terrestre, a la força de fregament atmosfèric i a la gravetat de la Lluna i el Sol. S'estudien els efectes d'aquestes pertorbacions i també com realitzar l'implementació mitjançant el Jet Transport. El capítol acaba amb algunes simulacions de Monte Carlo per extreure informació d'una col·lisió semblant a la produïda entre els satèl·lits Iridium-33 i el Kosmos-2251 l'any 2009.

- L'annex A explica breument les funcions desenvolupades per a aquesta tesi i s'introdueixen unes petites notes sobre paral·lelització de codis en C mitjançant openMP.

Contents

Chapter 1

Introduction

As time goes by, all what surrounds us changes. From the ancient times, the human being has studied those changes. The curiosity to enlarge the knowledge of what surrounds us (either the earth that we step, the air that we breathe or the space that we see) characterises the humanity. Astronomy is one of those ancient sciences that studies how things evolve with time, in this case the celestial bodies. The first astronomers were worried about what they could see in the starred dark nights; they were asking why all those white points were moving in the sky and which was the origin of their light.

After years of observations, the astronomers of those ancient cultures, perhaps motivated by mystical reasons, were able to predict and know the movements of the planets. Some of them, like the Mayan astronomers, were able even to predict Solar and Lunar eclipses. Long series of eclipses observed and predicted by the Mayans have arrived to our days.

Nowadays our knowledge of the space has improved from those tables of eclipses; we know what the stars are, of what they are made of and even which laws rule the movement of the different celestial bodies. Since the seventeenth century, mathematical formulations for the movement of the planets have been developed, and important names of the science history appear in these studies such as Johannes Kepler, Isaac Newton or Albert Einstein, just to name some of them.

This thesis tries to put a little grain of sand on the building constructed by those giants on whose shoulders we are standing: the science building. Inside the building of science there are many disciplines and areas of knowledge. One of them is Dynamical Systems Theory, in which this thesis can be fitted. Dynamical Systems Theory is the branch of mathematics that studies the evolution with time of a certain measurable quantity that varies under some rule of movement or change. The rule can express the change in a continuous or a discrete way. In the first case, we talk about continuous dynamical systems, this kind of systems studies the evolution of the observable at each infinitesimal instant of time. In the second case we talk about discrete dynamical systems, this kind of systems study the evolution from time to time, for instance which will be the observed state at each minute, day or century.

The objective of the Dynamical Systems Theory is to study how the initial states of a system evolve from an initial time to a final one. In some problems, it is possible to determine some structure which helps to separate the dynamics in different regimes. These structures act as barriers for the states and forces the movement from certain regions to others. Therefore, the obtention of these structures determines how a given state changes as time evolves, we talk then about dynamical transport. The aim of this thesis is to explain which are those dynamical transport mechanisms in some concrete problemes.

Coming back to Astronomy, and to the study of the motion of bodies in space, the states that we want to observe how they evolve with time are the position and velocity of a celestial body (or several of them) under the gravitational influence of at least one of them, together with other forces that perturbe the initial model.

The science that studies the laws of evolution, and what motions are produced under those laws, is Celestial Mechanics. Celestial Mechanics is based in Newton's laws of motion and Newton's law of universal gravitation. They are a good starting point to study how the stars, the planets, the moons and the asteroids

move around us. When we talk about the study of the movement of spacecraft, their trajectories and their manoeuvres, we are talking about Astrodynamics or Orbital Dynamics.

Around this kind of problems several branches of science are merged. We can find working on them, physicists, aerospace engineers, and mathematicians, each one contributes with his own point of view and enriches the knowledge of this scientific area.

From a mathematical point of view, the problems that we study are described by a continuous dynamical system. Its evolution laws are given by means of differential equations on a given manifold, in this case $\mathbb{R}^n$.

Let us assume the following general setting. An ordinary differential equation,

$$\dot{\vec{x}} = f(t, \vec{x}), \tag{1.1}$$

describes the motion of the observables (in our case, positions and velocities of celestial bodies) that we want to study. The function f gives the direction in which a particle located at $\vec{x} \in \mathbb{R}^n$ will move at the instant of time t, that is, the vector field of the directions of the states. It describes a flow map,

$$\begin{array}{rcl} \phi : \mathbb{R} \times \mathbb{R} \times \mathcal{D} & \longrightarrow & \mathcal{D} \\ (t, t_0, \vec{x}_0) & \longmapsto & \phi(t; t_0, \vec{x}_0) = \vec{x}_t. \end{array}$$

which returns the position $\vec{x}_t$, at the time t, of a particle that at time t_0 was located at $\vec{x}_0$.

The explicit solution of this kind of equations is, in general, not known. It is possible to explicitly find $\phi(t; t_0, x_0)$ in some simple cases. However, for most of the problems that we may face it will not be possible.

When it is not possible to compute the explicit solution, it is still possible to compute an approximation of ϕ using a numerical method. There is a huge variety of numerical methods to integrate ordinary differential equations. The most simple one is Euler's method. It uses the first order approximation of ϕ, using the vector-field f, to obtain the first derivatives. Then, for a small time interval, δ, the method gives the approximate position at time $t_0 + \delta$. To obtain an approximate solution of ϕ at other values of t, the method must be iterated until the reached time is the appropriate one. If one looks for greater accuracy, higher order implementations are possible. A well known family of this higher order methods is the Runge-Kutta (RK) family, developed by C. Runge and M. W. Kutta around 1900. These methods use an iterative algorithm to compute the higher order approximations of the solution ϕ. A popular method of this family is the *classical Runge-Kutta* of order 4 (RK-4). For fixed order methods a constant step-size must be chosen, which has two main inconveniences: either the final aproximation is not accurate enough because the step size is too large (this can happen when the solution goes close to a singularity), or when the step size is taken to small, in order to maintain the accuracy, long CPU computations are required. To avoid these drawbacks, E. Fehlberg developed the so called RK-Fehlberg methods. These methods use two consecutive orders to estimate the time-step size, in this way when the solution is in a regular region the time-step is enlarged, while when the solution is close to a singularity the time-step is reduced. A popular version of the RK-Fehlberg method is the one using orders 7 and 8 for the step control; this method will be denoted as RK-78 in this thesis.

All the integration methods commented use iteratively closed formulas to obtain a polynomial expansion of the flow map. If one wants to use higher orders, the formulae have to be obtained again. This can be an specially hard work if the orders must be very high. To avoid the computations of the high order terms, it is possible to use Taylor's methods for the integration of ODEs. These methods where already known and used by Newton and Euler to solve differential equations by means of infinite series. The idea is to use automatic differentiation to obtain the high order terms; in this way, the coefficients of the expansion can be determined without computing any specific formulae.

Using Taylor's methods we are able to obtain time expansions of the flow. In some situations we may also be interested in phase space expansions of the flow. For instance when we want to know how a small variations on the initial conditions affect the final ones. This leads us to talk about what is known as variational equations. Assume that we are interested in knowing the position at time t of an initial condition $\vec{x}_0 + \vec{\xi}$ with an undetermined $\vec{\xi}$. To solve this question, one possibility is to obtain a first order Taylor expansion of the solution:

$$\phi(t; t_0, \vec{x}_0 + \vec{\xi}) = \phi(t; t_0, \vec{x}_0) + D_{\vec{x}_0}\phi(t; t_0, \vec{x}_0)\vec{\xi},$$

where the matrix $J = D_{\vec{x}_0}\phi(t; t_0, \vec{x}_0)$ can be computed solving the linear system of differential equations:

$$\begin{cases} \dot{J}(t) = D_x f(t, x) J(t), \\ J(0) = Id. \end{cases}$$

Observe that, in this way, the first order Taylor expansion of the flow is obtained. If better approximations are needed, it is possible to use higher order expansions of the flow. The problem arises in how to compute the coefficients of these higher order expansions. To obtain the first order coefficients we differentiated with respect to time in the vector field. The classical methods to obtain the higher order variational terms consist in derivating, as many times as required, the vector field 1.1 until the appropriate order is reached. However, these methods require a considerable handwork before the final equations are obtained, and probably some errors will appear in the resulting formulae. In Chapter 2 we explain an alternative technique to avoid the handwork of computing the high order coefficients. The technique is known as *Jet Transport*, and also as Differential Algebra in some literature. The fundamental idea of the Jet Transport is similar to the one of Taylor's method: expand the solutions in Taylor series using automatic differentiation. Now, instead of using time as the independent variable to compute the expansions, the phase space variables are used. Therefore, at each time t the solution is expressed by means of a polynomial in l variables,

$$P(\xi_1, \ldots, \xi_l) = \sum a_{k_1,\ldots,k_l} \xi_1^{k_1} \cdots \xi_l^{k_l},$$

that, when it is evaluated at $\vec{\xi} = (\xi_1, \ldots, \xi_l)$, returns an approximation of the state $\phi(t; t_0, \vec{x}_0 + \vec{\xi})$.

To simplify the notation multi-indices will be used. A *multi-index* k is a vector of natural numbers, $k \in \mathbb{N}^n$ that designates the exponents of a given monomial. In that way we shorten the notation making it more compact. The monomial $a_{k_1,\ldots,k_l} \xi_1^{k_1} \cdots \xi_l^{k_l}$ can be written as $a_k \xi^k$.

Through all the thesis the multi-index notation is used extensively, and arrows, such as $\vec{k}$, will never be used to denote a multi-index. The *order* of a multi-index k, $|k|$, is the defined as:

$$|k| = k_1 + \cdots + k_l$$

In the development of some formulae, it is useful to define an special order inside the multi-index: the *order by components*. Given two multi-indices, k and $\tilde{k}$, it is said that k is smaller or equal by components than $\tilde{k}$, $k \leq_c \tilde{k}$, if $k_i \leq \tilde{k}_i$ for $i = 1, \ldots, l$. The inequality is strict, $k <_c \tilde{k}$, if $k_i < \tilde{k}_i$ for $i = 1, \ldots, l$. Observe that this is an order relation but is not a total order one. For instance, if we consider $k = (1, 2)$ and $\tilde{k} = (2, 1)$, then neither $k \leq_c \tilde{k}$ nor $\tilde{k} \leq_c k$. Intuitively, if $k <_c \tilde{k}$ then there exist a multi-index $\overline{k} \neq 0$ such that $k + \overline{k} = \tilde{k}$.

All along the thesis, the letter k will be reserved for multi-indices, although, in some cases, other letters will be used to denote them.

With this multi-index notation, the degree n of a monomial is defined as the order of the multi-index. The monomials are join together in truncated series. The order of a truncated series and, for extension, the order of a truncated Taylor expansion, is the maximum degree of its monomials.

Most of the systems studied in this thesis are multi-dimensional. For this reason, also the vectorial notation is used. The i-th component of a vector, either a multi-index or an element of $\mathbb{R}^n$ is designated by the sub-index i. However, there are some cases where the use of a sub-index may be confusing, for instance when the vector already has other sub-indices or when we want to select the i-th component of a vector given by an operation. In this situations a right square bracket ( ]) is added before the sub-index. For instance, given a matrix A and a vector $\vec{x}$, $A\vec{x}]_i$ designates the i-th component of the product $A\vec{x}$.

1.1 Dynamical indicators

Once we know how initial conditions can be properly propagated an in a vector-field, we are also interested in how to extract as much information as possible about the global dynamics associated to the vector-field.

The first information that can be extracted from a dynamical system is from the study of its fixed points, i.e. those points $\vec{x}$ that verifies $f(\vec{x}) = 0$. This are the stationary points of the system and represent those

states that never move with time. The fixed points can be classified as a function of what the nearby points do. A fixed point is said to be stable if its neighbouring points remain close to it for all time. A fixed point is said to be unstable if there is some neighbouring point that moves far away from it. This classification can be characterised using the eigen-values of the differential matrix of the vector-field at the fixed point.

Let $D_{\vec{x}}f(t,\vec{x}_0)$ be the differential of the vector field at a fixed point x_0, λ_i the eigenvalues, and $\vec{v}_i$ the associated eigenvectors. Then, the fixed point is stable if the real part of all the eigenvalues is smaller than 0, i.e. $\Re\lambda_i < 0$ for $i = 1,\ldots,n$. The point is unstable if there exists i such that $\Re\lambda_i > 0$, in this case, an invariant manifold, tangent to the eigen-vector $\vec{v}_i$ at x_0, is associated to the fixed point. If the real part of the eigenvalue associated to that eigenvector is greater than 0, the invariant manifold tangent to $\vec{v}_i$ is unstable, and all the points on the invariant manifold will depart from the fixed point $\vec{x}_0$. On the other hand, if the real part of the eigen-value is smaller than 0, then the invariant manifold is stable and all the points on the invariant manifold approach the fixed point $\vec{x}_0$. If the real part of the eigenvalue is zero then the points in that direction do not approach nor move away from the fixed point. If the system is a real, then there will exist a second eigenvalue, conjugated to the first one, with zero real part. The eigenvectors of these two eigenvalues expand a plane where the solutions turn around the fixed point $\vec{x}_0$.

After studying the fixed points, the next step is to study the periodic orbits, or, more generally, the invariant tori. A periodic orbit is a solution of the vector-field $\dot{\vec{x}} = f(t,\vec{x})$ such that there exist a time T for which $\phi(t_0+T;t_0,\vec{x}_0) = \vec{x}_0$ for all $\vec{x}_0$ in the periodic orbit and all t. As it happen with the fixed points, the periodic orbits can also be classified as stable or unstable. The stable periodic orbits are characterized because the orbits around them remain close to the periodic orbit. However, the orbits around unstable periodic orbits move away from them. Similarly to the fixed points, the periodic orbits (and also the invariant tori) can have stable and/or unstable manifolds. All these elements define the skeleton of the dynamical system.

Both, fixed points and periodic orbits exhibit a regular behaviour; there are other kinds of solutions having also a regular behaviour, in the sense that all the points in a neighbourhood around them behave in a similar way. However, some systems can exhibit chaotic regions. Those regions can be recognised because two points in them, independently of how close they are, exhibit very different dynamical behaviors.

Distinguish between these chaotic and regular regions is important. For instance, we can be interested in knowing what happens when we deviate a little bit from our initial state: will be the orbit of the new point very close to the one of the initial point? or will they behave in very different ways?

Such kind of knowledge is of interest in Celestial Mechanics problems, the most classical one is to know if the solar system is stable or not. If we consider a planet orbiting a star, it is known that the planet will move in an elliptic orbit around it. However, when more planets are added to the system, their mutual gravitational attraction must be taken into account. Therefore, the orbit of the planets is not elliptic anymore. In this problem the question that we are interested in, is what will happen with those orbits, will they be far away from the elliptic orbits? or will they be almost elliptic?. An interesting reading on the history and the results of this problem can be found in [Las13].

A usual tool to study the regularity or chaoticity of the solutions of a dynamical system are the *dynamical indicators*. A dynamical indicator is an observable of the system that give us some information about the behaviour of its solutions. The Lyapunov exponents are one of such indicators. They where introduced by A. Lyapunov in 1892 as:

$$\sigma(\vec{x}_0) = \lim_{t\to\infty} \frac{1}{t} \log \|D\phi(t;t_0,\vec{x}_0)\|,$$

and have been widely used in the study of dynamical systems [VS15, VBS11, YN93].

Other dynamical indicators have been developed that increase the accuracy of the information provided, decrease the CPU time computation, or give alternative indications of similar properties. Some examples are: Finite Time Lyapunov exponent (FTLE), the Fast Lyapunov indicators developed by Froeschlé et al ([FGL97]), or MEGNO (Mean Exponential Growth factor of Nearby Orbits) developed by P.M. Cincotta and C. Simó ([CS00, CGS03]).

The FTLE have been largely used by G. Haller to establish the Lagrangian Coherent Structures (LCS) theory [HY00]. The LCS are structures that separate different kind of dynamical behaviours, playing a role similar to the one that the invariant manifolds do, however, they are not equivalent. Chapter 3 presents a

study of the computation of the FTLE and the LCS, and introduces some new dynamical indicators that exploit the information provided by the Jet Transport.

1.2 Application of the Jet Transport to collision avoidance

From the Astrodynamics point of view, the stability problem is also interesting. When a space agency, or satellite operator, launches a spacecraft, needs to know if the spacecraft will follow the foreseen nominal without performing station-keeping maneuvers, or some maneuvers will be required.

The motion in the Earth's neighbourhood is rich and complex. The movement of a spacecraft orbiting the Earth has different perturbations that are not negligible even for short periods of time. In addition, nowadays the Earth neighbourhood is populated by around 3 600 satellites and a huge number of space debris of different size; therefore, there is a risk of satellite collision, with either space debris or another satellites.

Possible collisions must be studied in order to see if it is worth to perform a collision avoidance manoeuvre and if after the execution of such manoeuvre the probability of collision, during a certain time interval, will remain high or not. Since the last decades of the twentieth century, scientists around the world have studied this problem. There are several options to deal with it. One option is to study the problem using analytical methods. The collision probability is obtained by computing the indefinite integral of an approximated probability density function, which is obtained by translating the initial error matrix to the final state [FH92, AA00]. The main drawback of the approach is that the propagation of the initial uncertainty matrix is done using a linear approximation and, therefore, the non-linear effects produce inaccuracies on the final result.

To deal with the non-linearities Monte Carlo simulations are used [dVP10, SBS$^+$11]. However, Monte Carlo methods require a high computational effort, since each set of initial conditions for the spacecrafts or debris involved in the collision must be integrated individually up to the collision (or minimum distance) epoch. As an alternative one can consider a Jet Transport evaluation. Some studies on that direction have been previously done by Morselli et al [MADLBZ15]. In Chapter 4 we study this problem. A discussion about how to deal with the different perturbations of the problem, as well as how the collision probability can be approximated combining Monte Carlo methods with the Jet Transport is done.

1.3 Dynamical test models

Along this thesis, different algorithms are developed either to test the Jet Transport machinery or to test the dynamical indicators developed. These basic tests models are introduced in what follows.

The first models are 2-dimensional mechanical systems, with 1 and $1 + 1/2$ degrees of freedom. They are easy to implement and, also important, to plot. These models are the simple pendulum and, since some of the applications developed are of interest for non-autonomous systems, a periodic perturbation on the simple pendulum.

The second models are related to Celestial Mechanics problems in dimension four (2 and $2 + 1/2$ degrees of freedom). They are the planar Circular Restricted Three Body Problem (CRTBP) and the planar Elliptic Restricted Three Body Problem (ERTBP).

1.3.1 One and one and one half degree of freedom systems

The first dynamical models considered are the simple pendulum and a time-periodic perturbation of it. These systems are a model for the motion of a body of mass m attached by a solid bar, of lengh l and negligible mass, to a fixed point. The only force acting on the body is gravity, i.e. there is no friction with the air neither with the joining points. Using dimensional units, the equations of motion are given by:

$$\ddot{x} = -\sin(x).$$

where x is the angle between the bar and the vertical axis. This system has two equilibrium points, both of them when the bar is completely vertical and at rest: one when the mass is below the bar, and the other one when the mass is above the bar. The first of these equilibrium positions is stable, since a small perturbation of it will produce a small oscillation of the mass, but always remaining close to the equilibrium position. The second one is unstable, any small perturbation makes the pendulum to fall, going through the bottom position and going up to a position close to the vertical one where it started.

In the pendulum it is possible to distinguish between three different motions. The librations, the circulations and the asymptotic motions. The *librations* are the oscillations of the pendulum around its lower equilibrium position, and they can be characterised by a maximum angle ($< \pi$) of deviation with respect to the stable equilibrium. The *circulating motions* happen when the energy of pendulum is large enough to keep it rotating in the same direction, clockwise or counterclockwise, for ever. The asymptotic motions, also known as infinite time motions, are between these two. The pendulum starts its motion at the above vertical position and takes infinite time to do one and only one revolution. Figure 1.1 shows the three kinds of motion in the phase space of the pendulum.

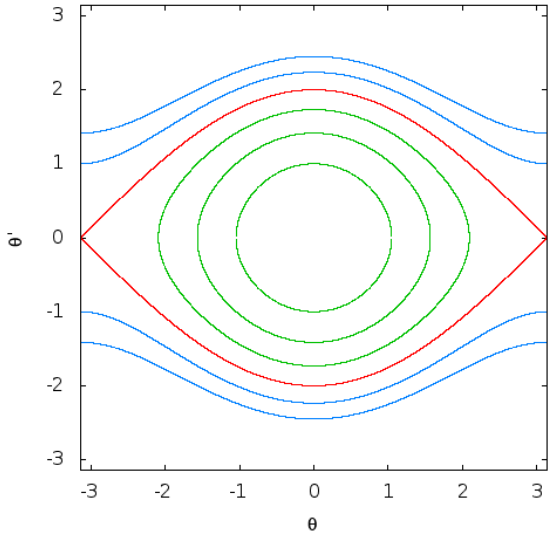


Figure 1.1: The figure shows the phase space of the simple pendulum with the different kinds of motion: librations (green), circulations (blue), and asymptotic motions (red).

The two asymptotic orbits correspond to the hyperbolic invariant manifolds of the unstable fixed point, and are also known as separatices, because the separate the oscillations from the circulations. Observe that all the orbits of the pendulum are periodic: for each orbit there exist a time T such that the motion is repeated after each T time units.

Some of the tools that have been developed have been designed to study also non-autonomous systems, i.e. those systems in which the vector-field depends explicitly on time. As a test example for these system, a periodic perturbation of the simple pendulum has been used. The equations of motion considered for this problem are:

$$\ddot{x} = (2.5 \cos 5t - 1) \sin(x).$$

This equation simulates the addition of a power source to the system that impulses the pendulum in a sinusoidal way.

The dynamics in this situation becomes more complex. Both fixed points remain, however most of the orbits around them that were periodic in the simple pendulum case, are not periodic anymore. Now, only remain periodic some orbits with period multiple of the period of the perturbation $T_p = 2\pi/5$. The two regimes (librations and circulations) still exists, but the frontier between them is not so clear as before.

The invariant manifolds that in the unperturbed case connect the unstable point, now form an homoclinic tangle. It can be represented by looking at the invariant manifolds of the stroboscopic map of time T_p. Observe that this map has the fixed points of the system as fixed points, but in addition it also has as fixed points the points corresponding to the periodic orbits of period T_p. Figure 1.2 shows the invariant manifolds for the previously unstable periodic point, together with the intersection of the invariant manifolds of two periodic orbits, above and under the libration zone.

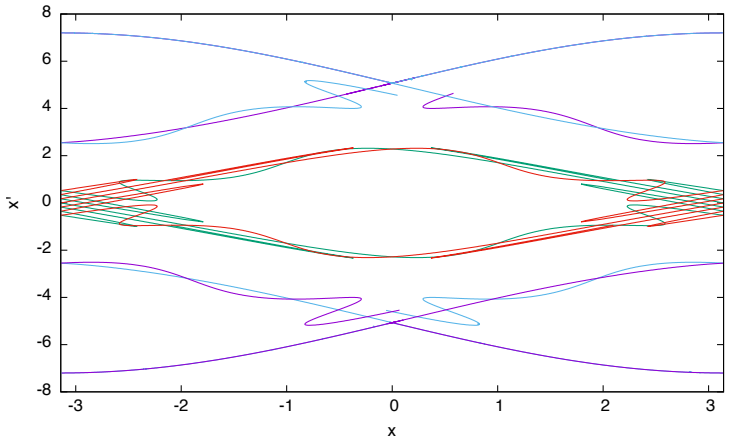


Figure 1.2: For the perturbed pendulum, stable and unstable invariant manifolds of the stroboscopic map of time T_p, defined by the period of the perturbation, for different fixed points. The central curves are the stable (green) and unstable (red) invariant manifolds of the unstable fixed point. The top and bottom curves are the stable (blue) and unstable (purple) manifolds of the fixed point of the stroboscopic map given by the periodic orbit of period T_p.

Observe that the bottom most and upper most stable and unstable manifolds seem to coincide. However, this is not the case, there is a very small difference than cannot be appreciated in the plot. Under generic perturbations, the splitting of the two homoclinic connections does not need to be the same. For the homoclinic connection close to the libration zone, the splitting of the manifolds is larger than the other one. For more information about this problem see [SV09].

1.3.2 The Restricted Three Body Problems

The two last test models considered are the planar Circular Restricted Three Body Problem (CRTBP or CR3BP) and the planar Elliptic Restricted Three Body Problem (ERTBP or ER3BP).

The CR3BP studies the motion of a massless particle M, for instance a spacecraft or an asteroid, under the gravitational attraction of two massive bodies (primaries) M_1 and M_2, of masses m_1 and m_2 respectively, moving in circular orbits around their centre of mass. Using the appropriate dimensionless units, it is possible to reduce the number of parameters of the problem to only one, the so called mass parameter $\mu = m_2/(m_1 + m_2)$. The unit of distance is taken in such way that the two primaries are separated by a distance equal to one. The time unit is fixed in order that the two primaries do a revolution in 2π dimensionless time units. Finally, the mass of the two primaries is normalised to one, $m_1 + m_2 = 1$. Let $\mu = m_2/(m_1 + m_2)$ the dimensionless mass of the small primary, the mass of M_1 will be $1 - \mu$. The last step to obtain the equations of motion consist into consider the motion of all the bodies in a rotating coordinate system, in which the two primaries are fixed on the x-axis at $(\mu, 0)$ and $(\mu-1, 0)$. A more detailed description of the problem can be found in [Sze67]. Let us remark some basic characteristics of the model.

Figure 1.3 shows the location of the distribution of the masses. The mass M is the studied one.

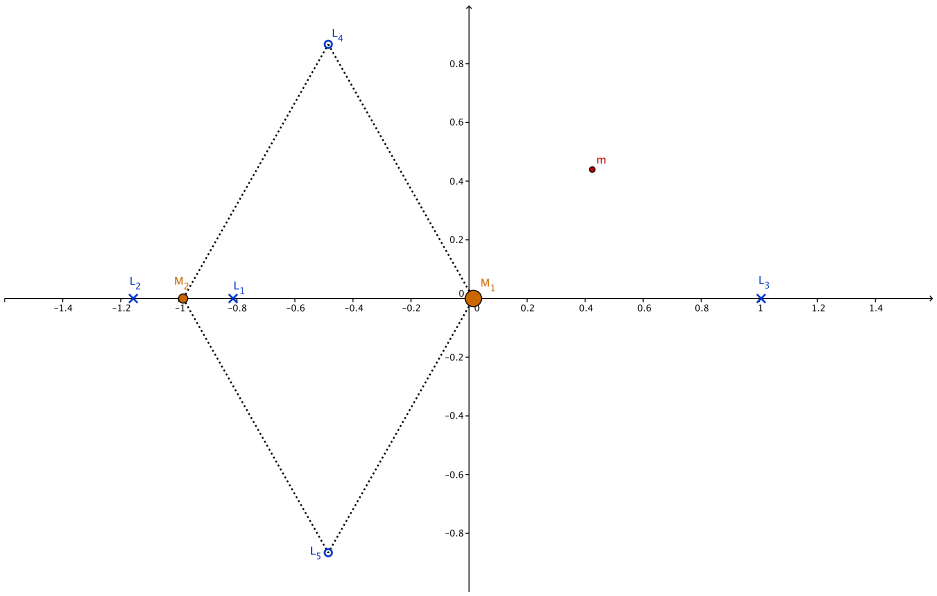


Figure 1.3: Location of the primaries M_1 and M_2 in the rotating dimensionless coordinate system (synodic reference system). The Lagrange points are also plotted in the figure.

Under the the above considerations, the equations of motion are:

$$\begin{aligned} \ddot{x} - 2\dot{y} &= \Omega_x, \\ \ddot{y} + 2\dot{x} &= \Omega_y, \end{aligned} \tag{1.2}$$

where $\Omega(x,y) = (x^2+y^2)/2 + \mu/r_1 + (1-\mu)/r_2 + \mu(1-\mu)/2$, $r_1 = \sqrt{(x+1-\mu)^2+y^2}$, $r_2 = \sqrt{(x-\mu)^2+y^2}$.

The above equations have a first integral, that can be written as:

$$E(x,y,\dot{x},\dot{y}) = \frac{1}{2}(\dot{x}^2+\dot{y}^2) - \Omega(x,y).$$

In some cases instead of the "energy" E, the Jacobian integral defined as $C_J = -2E(x,y,\dot{x},\dot{y})$, is used. The value of C_J is known as the Jacobi's constant. For some levels of the energy, i.e. for some fixed values of the first integral, there are regions of the configuration space (x,y) that are not attainable. This forbidden regions are called Hill regions, $\mathcal{H}_e$, and are defined as:

$$\mathcal{H}_e = \{(x,y) | \Omega(x,y) + e > 0\}$$

For small values of the energy $E = -C_J/2$ Hill's regions enclose the two primaries and do not allow any movement between them. When we increase the value of the energy, a small gap in Hill's region appears that allows the transition between the two primaries. However, it is still not possible to leave the region around both masses or to enter it. For larger values of the energy a new gap appears in Hill's region that allows to enter and exit the inner regions. Then a third gap is open. Finally, for big values of the energy all the configuration space is attainable. Figure 1.4 shows the evolution of Hill's regions with C_J.

We have seen that, solving the variational equations, we can compute how small perturbations in the initial conditions affect the final position of the small body M. In the CR3BP, the variational equations can be written as:

$$\dot{A} = DF\, A,$$

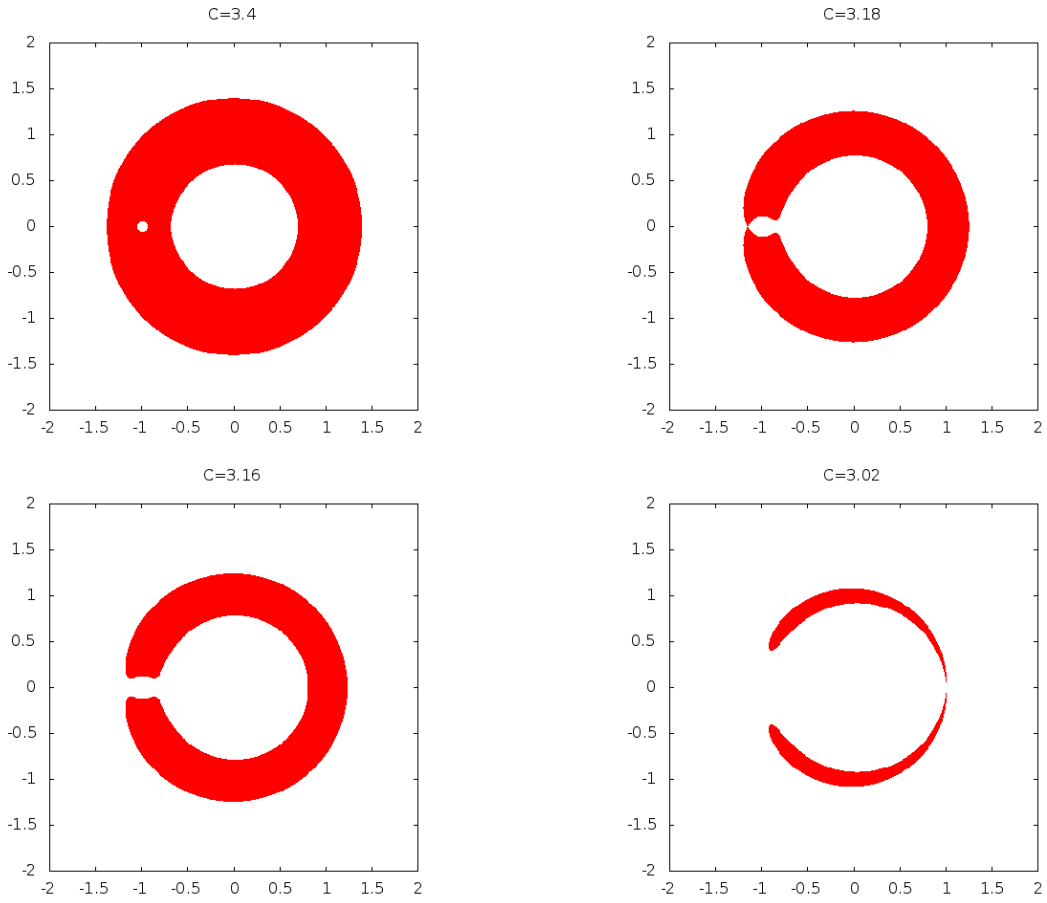


Figure 1.4: Hill's regions for different values of the Jacobi constant. From left to right and up to bottom: $\mathcal{H}_{-2}$, $\mathcal{H}_{-1.509}$, $\mathcal{H}_{-1.508}$ and $\mathcal{H}_{-1.501}$.

where $A = (a_{ij})$,

$$DF = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \Omega_{xx} & \Omega_{yx} & 2 & 0 \\ \Omega_{xy} & \Omega_{yy} & 0 & 2 \end{pmatrix},$$

and

$$\begin{aligned}
\Omega_{xx} &= \frac{3(1-\mu)(x-\mu)^2}{\left(y^2+(x-\mu)^2\right)^{5/2}} - \frac{1-\mu}{\left(y^2+(x-\mu)^2\right)^{3/2}} + \frac{3(x-\mu+1)^2\mu}{\left(y^2+(x-\mu+1)^2\right)^{5/2}} - \frac{\mu}{\left(y^2+(x-\mu+1)^2\right)^{3/2}} + 1, \\
\Omega_{xy} &= \frac{3y(1-\mu)(x-\mu)}{\left(y^2+(x-\mu)^2\right)^{5/2}} + \frac{3y(x-\mu+1)\mu}{\left(y^2+(x-\mu+1)^2\right)^{5/2}}, \\
\Omega_{yx} &= \frac{3y(1-\mu)(x-\mu)}{\left(y^2+(x-\mu)^2\right)^{5/2}} + \frac{3y(x-\mu+1)\mu}{\left(y^2+(x-\mu+1)^2\right)^{5/2}}, \\
\Omega_{yy} &= \frac{3(1-\mu)y^2}{\left(y^2+(x-\mu)^2\right)^{5/2}} + \frac{3\mu y^2}{\left(y^2+(x-\mu+1)^2\right)^{5/2}} - \frac{1-\mu}{\left(y^2+(x-\mu)^2\right)^{3/2}} - \frac{\mu}{\left(y^2+(x-\mu+1)^2\right)^{3/2}} + 1.
\end{aligned}$$

The variational equations are used in Chapter 2 to test the correctness of the computations done using

the Jet Transport, and in Chapter 3 to determine some dynamic indicators.

The CR3BP (1.2) has five equilibrium points. Three of them are collinear with the primaries and its position $(x, 0)$ can be found solving a fifth-order equation, for instance:

$$x^5 + (3-\mu)x^4 + (3-2\mu)x^3 - \mu x^2 - 2\mu x - \mu = 0,$$

to determine the position of L_2. Similar equations are solved to determine L_1 and L_3. This equilibrium points exhibit, for any value of μ, a saddle $\times$ centre behaviour.

The other two fixed points are located in the third vertex of an equilateral triangle formed with the two primaries. For a small values of the mass parameter μ, they are of centre $\times$ centre type. Figure 1.3 also shows the location of the fixed points in the rotating dimensionless coordinate system.

Due to the centre part of the collinear points, there are periodic orbits around them, the so called Lyapunov periodic orbits, and thanks to the saddle component these periodic orbits have stable and unstable invariant manifolds. Figure 1.5 shows a representation of the invariant manifolds of a Lyapunov periodic orbit around L_1. A study of the Lagrange points and the phase space around them can be found in [GLMS85, GLMS87, GJMS91, GJMS93].

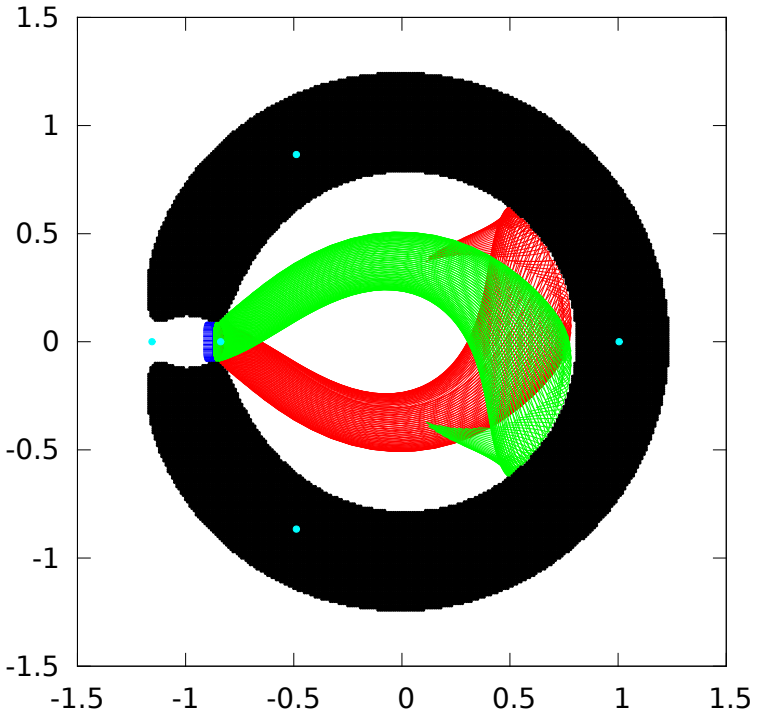


Figure 1.5: Lyapunov periodic orbit (dark blue) and the stable (green) and unstable (red) invariant manifolds departing from it. The Hill's region for the corresponding value of the energy (black) and the fixed points (cyan) are also represented.

In Chapter 3 we will show how some of the dynamical indicators introduced can detect the invariant manifolds associated to the unstable Lyapunov periodic orbits. The detection of already known structures is not the main objective of these indicators, neither the already existing indicators nor the new ones. Their objective is to detect structures in non-autonomous systems. For them it is not straightforward to compute the invariant manifolds. However, we will see that using the indicators we can capture the substitutes of the invariant manifolds in such cases.

The Elliptic Restricted Three Body Problem (ER3BP) studies the motion of a massless object within the gravitational potential field generated by a two-body system following elliptical orbits around the centre of mass. To maintain the position of the primaries fixed in a rotating reference frame, a periodic pulsation is added to the synodic system obtained for the CR3BP with the resulting equations:

$$\begin{aligned} x'' - 2y' &= \frac{1}{1+e\cos(f)}\Omega_x, \\ y'' + 2x' &= \frac{1}{1+e\cos(f)}\Omega_y, \end{aligned} \tag{1.3}$$

where e is the eccentricity of the ellipse of the primaries, f is the true anomaly, which now substitutes time as independent variable, and the prime denotes the derivative with respect to f. When the eccentricity is equal to 0 the problem becomes the CR3BP. When $e > 0$, E is no longer a first integral; despite that, one can use a pseudo-energy:

$$\hat{E}(x, y, x', y') = \frac{1}{2}(x'^2 + y'^2) - \frac{\Omega(x, y)}{1+e\cos(f)},$$

in order to do some reductions of the dimension of the system.

1.4 Structure of the thesis

This thesis has three main chapters whose contents is briefly described in what follows.

- In Chapter 2 the main tool used in the thesis is introduced, the Jet Transport. To use it a polynomial algebra must be implemented. The chapter starts by explaining how to adapt the polynomial algebra to work in a computer environment. The first section is addressed to the efficient storage of polynomials. Then the basic operations with polynomials follows (the product by an scalar, the addition, the product and the division of two polynomials) There are also introduced elementary operations like the exponential, the logarithm, sines and cosines, as well as the derivation and integration of polynomials. The following sections discusses some elaborated routines like the propagation of flow maps, including the computation of the Poincaré maps, and some techniques to improve the results obtained, the functional inversion of polynomials and the mapping of densities.

- In Chapter 3 some dynamical indicators are introduced. First a revision of the finite time Lyapunov exponents and the Lagrangian Coherent Structures is done. Some algorithms are implemented to reduce the high computational costs that they require. Then, the Jet Transport is applied to determine alternative indicators of the dynamics. Four new indicators are introduced, the maximum initial box, the maximum expansion rate, the maximum contraction rate and the expansion rate on the normal space, a small variation of the maximum expansion rate (see Section 3.3 for more information). Next, an structure extraction algorithm is developed to summarise the information given by the dynamical indicators. The Chapter finishes with an application of the dynamical indicators to the detection of practical stability zones on the CR3BP.

- In Chapter 4 the collision between satellites is studied. The Chapter starts with the study of the perturbations that affect the motion of a satellite around the Earth. A two body problem is considered with perturbations coming from the Earth potential, the atmospheric drag and the gravitational forces produced by a third body, in that case the Sun and the Moon. The effects of these perturbations are studied, as well as how to implement them in a Jet Transport procedure. The Chapter finishes with some Monte Carlo simulations to extract data in a collision similar to the Iridium-Kosmos collision produced in 2009.

- Appendix A contains a summary of the most important functions developed during the project, as well as some notes on the parallelization of the C code using openMP.

Chapter 2

Jet Transport

The present chapter is devoted to introduce and develop an algebra of operations with polynomials, as well as the set of techniques, based on them, known as *Jet Transport*. The operations of the algebra take one or several polynomials and return a new polynomial as the result of applying the implemented operations. The four basic arithmetic operations, addition, subtraction, multiplication and division, are considered. More complicated functions, like powers, exponentials, logarithms or trigonometric functions, are also included as operations of the algebra. Even more, it is possible to include some algorithms as operations, for instance to solve a system of equations or to propagate the neighbourhood of an initial condition through the flow of a vector field. These last operations are the ones known as Jet Transport (recall that the jet of a function is its truncated Taylor expansion).

Using this kind of operations we are able to determine the Taylor expansion of a function specified by a computer program, and, therefore, the derivatives of such function. Due to this reason, polynomial algebra is also known as *automatic differentiation* or *differential algebra*. One of the first differential algebra implementations was *COSY Infinity*, developed and implemented by Martin Berz and Kioko Makino in the late 90's [Be07] to study beam accelerators. Later, several applications have been developed and applied to other fields, in particular in astronomy and astrodynamic problems [ADLBZB10, VAdLL14, WDLA$^+$15, MADLBZ15].

Aside from the already mentioned COSY Infinity, there are other polynomial algebra packages, such as *TIDES*, developed by Abad et al.[ABBR12], or the *CAPD DynSys Library*, developed by D. Wilczak [Wil15]. The polynomial algebra used in this thesis has been completely written and developed by us.

All the differential algebra implementations are based on the following idea: given a function f one can obtain a local power series representation of f around a point a, denoted as $P_f(\xi)$, by means of its Taylor expansion:

$$P_f(\xi) = \sum_i \frac{f^{(i)}(a)}{i!}\xi^i = \sum_i \frac{f^{(i)}(a)}{i!}(x-a)^i,$$

therefore, any function can be considered as an element of the polynomial algebra setting through its Taylor polynomial. In general, to represent any (non degenerate) function f with exact precision, an infinite number of coefficients are needed and, since it is not possible to store all this information in a computer, what is done in practice is to implement a truncated polynomial algebra of order n.

In what follows, $\mathcal{P}(\vec{\xi})$ will denote the algebra of polynomials in $\vec{\xi}$, and $\mathcal{P}_n(\vec{\xi}) \subset \mathcal{P}(\vec{\xi})$ the polynomial algebra of truncated polynomials up to order n. Observe that, in general, $\vec{\xi} \in \mathbb{R}^l$, then the polynomial algebra will be designated by $\mathcal{P}^l(\vec{\xi})$ and the truncated polynomial algebra by $\mathcal{P}_{n,l}(\vec{\xi})$. In this case, $k = (k_1, ..., k_l)$ will be a multi-index and all the operations with them will be denoted, with no additional notation change, as if k was an integer.

The truncated polynomial algebra can be seen as the equivalent to the floating-point representation of real numbers. In the floating-point representation, any real number x is approximated by a rational number $\tilde{x}$, in the best possible way and using a given finite number of bits. Depending on the amount of memory

used for the storage, the representation will be better or worst. It must be noted that each $\tilde{x}$ can be the floating-point representation of several real numbers; for instance, assuming a floating point representation with three digits and base 10, $\tilde{x} = 0.333$ represents the numbers $x = \frac{1}{3}$, $x = 0.333$, as well as infinitely many numbers starting by 0.333.

The truncated polynomial algebra works in an analogous way to the floating-point representation of real numbers. Given a polynomial $P_f(\vec{\xi}) = \sum_k a_k \vec{\xi}^k \in \mathcal{P}(\vec{\xi})$, it is represented by $\tilde{P}_f(\vec{\xi}) = \sum_{0 \leq |k| \leq n} a_k \vec{\xi}^k$ in the truncated polynomial algebra $\mathcal{P}_n(\vec{\xi})$. For instance, in $\mathcal{P}_3(\xi)$, the polynomial $\tilde{P}(\xi) = \xi + \frac{\xi^3}{3!}$ represents the cubic polynomial $P(\xi) = \xi + \frac{\xi^3}{3!}$ as well as the function $\sin(\xi) = \xi + \frac{\xi^3}{3!} + o(\xi^5)$.

From now on, all the polynomials used will belong to a truncated polynomial algebra, for that reason, and in order to simplify the notation, the tilde in $\tilde{P}(\vec{\xi})$ will be removed.

When working with the truncated polynomial algebra, we are not representing a function in all its domain of definition but only in a given sub-domain, that depends on the behaviour of the function and the order of truncation of its Taylor series representation. As it will be shown, this is one of the main drawbacks, since after doing some operations the radius of validity, understood as the set of points where the series approximates well the value of a function, decreases fast. The only way to enlarge the convergence radius is to increase the truncation order, which may be very expensive in terms of CPU time.

Up to this point, we have considered the polynomial algebra from two different points of view. The first one is to consider their elements as just what they are –polynomials– and the second is to consider them as approximations of functions. There is a third one, which is the set point-of-view. An function f defined in the real numbers transforms a point x_0 into another one $f(x_0)$. With a polynomial algebra one can transform a set of points U into another set $f(U)$. To do it, consider U parametrised by the polynomial $P(\vec{\xi}) = \vec{x}_0 + \vec{\xi}$, then, with the rules of the algebra, apply f to $P(\vec{\xi})$ to get $Q(\vec{\xi}) = f(P(\vec{\xi})) = f(\vec{x}_0) + \sum_k b_k \vec{\xi}^k$. In this way, given a point $\vec{y} \in U$, one can compute the deviation $\delta(\vec{y}) = \vec{y} - \vec{x}_0$ and determine $f(\vec{y})$ computing $Q(\delta(\vec{y}))$.

In order to work with the polynomial algebra one may think that the best way to compute a certain function f (i.e. a trigonometric function) of some polynomial $P(\vec{\xi})$ is to compute first the Taylor expansion of f and after do the composition of both polynomials. In general, a much better option is to develop some iterative formulae to obtain the coefficients of the polynomial $f(P(\vec{\xi}))$. In this way, the computation of a given function in the polynomial algebra is not done using composition rules but using built-in functions that perform all the required computations.

The key point of any polynomial algebra is how the polynomial coefficients are stored. Any operation done with polynomials will require to locate some input coefficients, as well as to determine where the output coefficients must be stored. Therefore, it is worth to spend some time to decide and construct a good storage and retrieval system. For this reason, the first section of this chapter is devoted to discuss some of the storage systems that are available together with the one that has been selected. The subsequent sections are devoted to introduce the different functions and tools that will also be used.

The second section introduces the basic operations, like the addition, subtraction, product and division, together with some elementary functions like the exponential, logarithm, sine and cosine. The section ends with some time and efficiency tests of the implemented functions.

The next section is devoted to introduce the propagation through flows of vector fields. First, Taylor's method, for integration of ordinary differential equations, is presented as a jet in the independent variable. Next, Taylor's method is modified in order to accept polynomials as the coefficients of $\vec{\xi}$. The power of the Jet Transport methodology will be illustrated with several examples.

The fourth and fifth sections explain two tools frequently used in flow propagation. The first one is related to the computation of Poincaré maps and will be used to determine when and where some points will reach a certain surface of section defined, for instance, by $x = const.$ The second one gives some possible solutions to one of the drawbacks of the truncated polynomial algebra: the radius of validity, this is, where the computed polynomial approximates well the true solution of the problem. Two possible solutions are given: the first one splits the initial propagated region changing the initial polynomials, while the second option splits the propagated region at some point during the integration.

The sixth section of the chapter introduces three different methods to compute the polynomial P that approximates the inverse of a function f, this is, to determine the polynomial $Q(\vec{\xi}) \approx f^{-1}(\vec{\xi})$ such that

$P(Q(\vec{\xi})) = Q(P(\vec{\xi})) = \vec{\xi}$. This will be a useful tool in many problems. Finally, the last section introduces how to map a probability density function from an initial point to a final one using a mapping f. That mapping can be, for instance, the flow map after t time units. This tool will be applied in the last chapter to compute the probability of impact of satellites.

2.1 Storing polynomials

The storage of polynomials is the basis of any implementation of a truncated polynomial algebra; a good storage procedure will make the difference with other polynomial algebra implementations. Any operation or function with polynomials will need to use their coefficients, the faster the access is the faster the operations will be done. The storage and retrieval is usually the bottleneck of the CPU time computations; for instance, in the computation of the product of two polynomials, the CPU time required by the functions that determine which are the coefficients that must be used can be more than one half of the total CPU time. Therefore, as it has already been said, this is one of the basic ingredients of any polynomial algebra implementation.

The simplest storage procedure is the one of polynomials in one variable ξ and degree n: $P(\xi) = \sum_{i=0}^{n} a_i \xi^i$. With this kind of polynomials the storage is easy since we only need to use an array of dimension $n+1$ and store at each component of the array the corresponding coefficient a_i.

Another kind of polynomials easy to store are the polynomials with l variables and degree one. For these polynomials we can also use an array, of dimension $l+1$, to store its coefficients. The first component of the array stores the zero-order coefficient and the remaining ones the coefficient of the l variables; in this way, the polynomial $P(\vec{\xi}) = a_0 + a_1\xi_1 + a_2\xi_2 + \cdots + a_l\xi_l$ is stored in the array $a = [a_0, a_1, a_2, \ldots, a_l]$.

These are two simple cases for which is easy to find an efficient storage procedure. When the number of variables and order increases the situation gets more difficult. We can still manage polynomials in two variables using a double array or a matrix. Each coefficient $a_{(i,j)}$ is stored in the i-th column j-row. In this way, the index of the column corresponds to the exponent of the first variable and the one of the row to the exponent of the second one. Observe that in this case we do not need to declare all the matrix but only a triangular one. For a polynomial of degree n, we need $n+1$ columns to keep information when the second variable is of order zero but n when the second variable is of order 1.

The extension of the above procedure is not useful when the number of variables and order of the polynomial are not known a priory. We are interested in storage procedures for an arbitrary number l of variables and any order n.

The selected storage procedure is the reverse lexicographic order with array access to the indices. This is a classical method implemented by other authors, for instance by À. Jorba in [Jor99].

Let $P(\vec{x})$ be a polynomial of degree n in l variables. We write $P(\vec{x}) = \sum_{i=0}^{n} P_i(\vec{x})$ with $P_i(\vec{x})$ homogeneous polynomials of degree i. The order of a polynomial expansion is the degree of the polynomial at which we truncate the expansion. The polynomial P can be written as $P(\vec{x}) = \sum_{|k|\leq n} a_k \vec{x}^k$. Observe that the multi-index notation has been used, so, if $k = (k_1, \ldots, k_l)$ then $|k| = k_1 + \cdots + k_l$ and $\vec{x}^k = x_1^{k_1} \ldots x_l^{k_l}$. To store a coefficient a_k of this polynomial an array with two indices $a[i][j]$ is defined according to the following rules:

1. The first index of the array gives the degree of monomial associated to a_k, in this way, the index i of $a[i][j]$ will be $i = |k|$. Therefore, the dimension of the first index is $n+1$.

2. The second index will gives the position of the monomial a_k inside the set monomials of degree $|k|$, after defining a certain order in this set. In this way, if the monomial with coefficient a_k has the position number r in the ordering, then $j = r$. According to this, the second index will have dimension

$$\psi_l(n) = \binom{n+l-1}{l-1} = \#\{k \in \mathbb{N}^l \text{ s.t. } |k| = n\}. \tag{2.1}$$

   The value of $\psi_l(n)$ can be computed a priory and stored in order to do not increase the computational time of the executions. Observe that, as it happens with the case of two variables, we do not need a square matrix for the storage, since we do not need the same memory for order 0 (only one coefficient) than for order n (which needs $\psi_l(n)$ coefficients).

The above procedure requires the determination of the place of the monomial a_k inside the set of monomials of order $|k| = n$, this is, in the set of multi-indices of the same degree $|k|$. The method selected for the ordering is the reverse lexicographic order. Given two multi-indices k and $\tilde{k}$ of the same order $|k| = |\tilde{k}| = n$, two numbers $\overline{k} = k_l k_{l-1} \dots k_1$ and $\overline{\tilde{k}} = \tilde{k}_l \tilde{k}_{l-1} \dots \tilde{k}_1$, in base $n+1$, are defined. Then:

$$k < \tilde{k} \iff \overline{k} < \overline{\tilde{k}}.$$

For instance, if $l = 4$, given the multi-indices $k = (5, 1, 4, 2)$ and $\tilde{k} = (5, 2, 3, 2)$, both of them of order $n = 13$, we compute the numbers, in base 14, $\overline{k} = 5142$ and $\overline{\tilde{k}} = 5232$. Since $\overline{k} < \overline{\tilde{k}}$, then:

$$\vec{\xi}^k = \xi_1^5 \xi_2^1 \xi_3^4 \xi_4^2 < \xi_1^5 \xi_2^2 \xi_3^3 \xi_4^2 = \xi^{\tilde{k}}.$$

Since the number of variables is not know a priory, the multi-indices associated to the coefficients are stored, using a base 2^q system, as follows:

$$\widehat{k} = k_2 + k_3 \cdot 2^q + \dots k_l \cdot 2^{(l-2)q},$$

where 2^q is the maximum order that can be stored in one variable. The value of q depends on how many bits are used. It can be computed dividing the amount of memory that we want to use to store all the multi-indices associated to the coefficients of a polynomial by, in principle, the number of variables l. For instance, using integers to store the index j of the ordering means to have 32 bits, if each variable uses the same number of bits, then $q = 32/(l-1)$ bits will be used for each variable. The -1 in the above denominator is due to the fact that the value of k_1 is not stored, since it can be recovered from:

$$k_1 = n - k_2 - \dots - k_l.$$

Since each bit can have two different values, the maximum exponent for a given variable will be $2^q - 1$.

For example, for $l = 4$, to store the multi-index $k = (5, 1, 4, 2)$ we have $q = 32/3 \approx 10$, therefore the maximum exponent will be 1023, and the representation of k in base 2^q is $1 + 4 \cdot 2^{10} + 2 \cdot 2^{20} = 2101249$.

Up to now, we have introduced four different ways to represent a multi-index:

1. The *vector* representation $(k_1, \dots, k_l)$, given by the vector of multi-index.
2. The *base* $n+1$ representation $\overline{k} = k_l k_{l-1} \dots k_1$.
3. The *base* 2^q representation $\widehat{k} = k_2 + k_3 \cdot 2^q + \dots k_l \cdot 2^{(l-2)q}$.
4. The *cardinal representation*, also called *code*, in which $(k_1, \dots, k_l)$ is represented by the place j that it has in the reverse lexicographic order.

In order to do operations, both the multi-indices and the associated code are the most useful representations. The base $n+1$ and base 2^q representations are mainly used as an intermediate notations to perform the transformations between the different representations. Next we explain how the transformations that have been implemented can be done.

- From vector to code representation.
  To compute the code of a multi-index $k = (k_1, \dots, k_l)$, the following quantities are needed:
  1. The number of multi-indices $(j_1, \dots, j_l)$ of length l (associated to polynomials with l variables) and order $|k|$ such that $0 \leq j_l < k_l$.
  2. The code of $(k_1, \dots, k_{l-1})$ considered as a multi-index of degree $n_l = |k| - k_l$, this is, the code of a monomial with $l-1$ variables and order n_l.

The first quantity can be computed as:

$$\psi_{l-1}(n_l+1)+\psi_{l-1}(n_l+2)+\cdots+\psi_{l-1}(|k|),$$

where ψ is defined in (2.1). The second we can computed applying recursively the first formula, since what is required is to know the position of a multi-index of degree n_l with $l-1$ variables, which is the same kind of computation. In this way, the final formula is:

$$\text{code} = \sum_{i=n_l+1}^{|k|} \psi_{l-1}(i) + \sum_{i=n_{l-1}+1}^{n_l} \psi_{l-2}(i) + \cdots + \sum_{i=n_2+1}^{n_3} \psi_1(i),$$

where $n_l = |k| - k_l$, and $n_i = n_{i+1} - k_i, i = 1, \ldots, l-1$.

- From code to base 2^q representation.
  To do this transformation an array is filled, such that for a certain degree and a code it gives the associated base 2^q number.
  The array is filled looking at all the multi-indices of degree n in reverse lexicographic order, starting with the first multi-index (which will have code 0) looking for the next one, and continuing until the end. The matrix is filled at the beginning of the process and used whenever it is needed to avoid additional computations.

- From base 2^q to multi-index representation.
  Given the 2^q representation $\widehat{k}$ of a multi-index, the the vector representation is computed according to:

$$\begin{aligned}
k_2 &= \widehat{k} \pmod{2^q},\\
\widehat{k}_2 &= \widehat{k}/2^q,\\
k_3 &= \widehat{k}_2 \pmod{2^q},\\
\widehat{k}_3 &= \widehat{k}/2^q,\\
&\vdots\\
k_l &= \widehat{k}_{l-1} \pmod{2^q},
\end{aligned}$$

  and finally $k_1 = |k| - \sum_{i=2}^{l} k_i$.

With the above transformations, given a multi-index we can obtain the code related to it directly, however, given a code we need first to do an intermediate transformation to obtain its multi-index representation.

The main advantage of the inverse lexicographic order is that when we add a new variable to the polynomial, the ordering of the initial polynomial does not change, since all the coefficients that involve the new variable are added at the end. In the inverse lexicographic order first appear the monomials of order n that involve only the first variable, after the monomials that also involve the second variable, later the ones that also involve the third and so on. The main drawback of this order is that is not straightforward to access a given coefficient.

Other options to store polynomials are available. A method that gives also good results is to store the polynomials using ordered trees. In this method a polynomial is saved by variables using a nested structure. The first element contains the independent term plus an array of links (pointers) to the same structure for the second variable. In that way we use the one variable storage at each moment. With this method a polynomial in l variables of order n is seen as a polynomial of order n in 1 variable which coefficients are polynomials of order n and $l-1$ variables. This storage procedure has advantages in the CPU time computations but its implementation is more complex. It has been used in [Har08, HCF+15].

2.2 The polynomial algebra

In this section we present the formulae that have been used to perform the basic operations and functions: addition, product, division, power, sines and cosines, exponential and logarithm. For the functions, one possibility that can be used is to first compute the Taylor expansion of the function and then do the composition of polynomials. However, as it has already been said, this is an inefficient method since it involves a large number of products in the second step (observe that at each composition we must compute the powers $P(\vec{\xi})^k$ and multiply the result by the corresponding coefficient). This is an expensive computation, even if it is done in a wise way.

The first part of this section is devoted to all the operations and functions, first we recall the well known formulae for the one variable case and after its generalisation to the case of multivariate polynomials. Next, some numerical results are given about the precision and speed of the procedures. Finally the last part introduces some auxiliary tools related with the Jet Transport. This tools are not direct operations of polynomials but are used to extract information of them. For instance, it is introduced how we can predict the maximal set of points that we can compute using a given jet without losing precision. Additionally, some notes about the evaluation of polynomials and their composition seen as evaluation of polynomials with polynomial variables, are also given.

2.2.1 Operations and functions with polynomials in several variables

The objective is to obtain iterative formulae to compute functions of polynomials. The formulae for the one variable polynomials can be found in the literature, see, for instance, the ones implemented by Jorba and Zou [JZ05] for the software package devoted to the numerical integration of ODEs using Taylor's method. For each operation, starting with the one variable case, we will show how to get the formulae for l variables.

Through all this section we will use two polynomials in l variables of degree n:

$$b(\vec{\xi}) = \sum_{k\in\mathbb{N}^l} b_k \vec{\xi}^k, \quad c(\vec{\xi}) = \sum_{k\in\mathbb{N}^l} c_k \vec{\xi}^k.$$

Product by scalars

One of the easiest and most used operations is the product by scalars. Let $\lambda \in \mathbb{R}$, given $b(\vec{\xi})$ we want to compute the polynomial $a(\vec{\xi}) = \lambda b(\vec{\xi})$. Clearly

$$a_k = \lambda b_k, \quad k \in \mathbb{N}^l.$$

Addition

The addition of two polynomials $b(\vec{\xi})$ and $c(\vec{\xi})$ is also one of the most frequent operations. Again it is done coefficient by coefficient, so, to compute $a(\vec{\xi}) = b(\vec{\xi}) + c(\vec{\xi})$ the following formulae are used:

$$a_k = b_k + c_k, \quad k \in \mathbb{N}^l.$$

Observe that the formulae for both the product by scalars and the addition do not depend on the number of variables and the order.

Product

The next important operation is the product of two polynomials. Let $p(\vec{\xi}) = b(\vec{\xi})\, c(\vec{\xi})$, with $p(\vec{\xi}) = \sum_{k\in\mathbb{N}^l} p_k \vec{\xi}^k$. In the case of only one variable, the Leibniz's formula gives the coefficients of the product as:

$$p_n = \sum_{i=0}^{n} b_{n-i} c_i.$$

In the case of severals variables, similar recurrent formulas can be obtained. Observe that each coefficient a_k will contain products of coefficients of b and c such that the sum of their exponents must be k, therefore, a general formula to determine p_k is:

$$p_k = \sum_{\substack{j \in \mathbb{N}^l \\ j \leq_c k}} b_{k-j} c_j,$$

where $j \leq_c k$ means that $j_i \leq k_i$ for $i = 1, 2, ..., l$.

In the implementation, instead of computing all the terms that contribute to a certain coefficient of p we compute all the products of monomials of b and c and then we add them to the corresponding coefficient of p. This is, for each pair b_k and c_j we determine the term of p to which the product contributes by adding both multi-indices $(k + j)$. Finally, the value of p_{k+j} is incremented by the product $b_k\, c_k$.
Assuming that at the beginning all the p_k are set equal to zero, and after computing all the products $b_k\, c_j$, the values of p_{j+k} are updated according to:

$$p_{j+k} = p_{j+k} + b_k\, c_j \quad \forall k, j \in \mathbb{N}^l, |k + j| \leq n.$$

Observe that the terms of $p(\vec{\xi})$ of degree greater than n are not computed, since the polynomials are truncated at order n.

The product of two polynomials is a basic operation that will appear in the following ones. All of them use the procedure just explained of updating the resulting polynomial using the appropriate coefficients of the two factors.

Division

To compute the polynomial result of the division of $b(\vec{\xi})$ and $c(\vec{\xi})$, $d(\vec{\xi}) = b(\vec{\xi})/c(\vec{\xi})$, we start writing this equation as:

$$d(\vec{\xi})\, c(\vec{\xi}) = b(\vec{\xi}), \tag{2.2}$$

In the case of only one variable, applying the Leibnitz formula, we get:

$$b_n = \sum_{i=0}^{n} d_{n-i} c_i = d_n c_0 + \sum_{i=1}^{n} d_{n-i} c_i,$$

from which we obtain

$$d_n = \frac{1}{c_0} \left(b_n - \sum_{i=1}^{n} d_{n-i} c_i \right).$$

Observe that in this equality the only unknown term is the d_n in the left hand side, all the terms in the right hand side are known, either because they have been computed previously or they are of one of the known polynomials b or c.

In the case of several variables, we can proceed in the same way. Looking those terms in (2.2) with multi-index k and matching both sides of the equality we obtain:

$$b_k = \sum_{\substack{j \in \mathbb{N}^l \\ j \leq_c k}} d_j\, c_{k-j} = d_k\, c_0 + \sum_{\substack{j \in \mathbb{N}^l \\ j <_c k}} d_j\, c_{k-j}. \tag{2.3}$$

Observe that in the sum of the last expression there is not any term coming from the polynomial d with degree k or higher. So we can isolate it to get:

$$d_k = \frac{1}{c_0} \left(b_k - \sum_{\substack{j \in \mathbb{N}^l \\ j <_c k}} d_j\, c_{k-j} \right) \quad k \in \mathbb{N}^l, |k| \leq n. \tag{2.4}$$

As it was done for the product, all the terms of order n are computed simultaneously, and after each product we look to which coefficient the term must be added instead of looking for all the terms that contribute to the coefficient k. The polynomial d is initialised with the same values than b, $d_k = b_k$, then, the product of all the pairs contributed to the degree $|k|$ are subtracted from the corresponding d_k:

$$d_k = d_k - d_j\, c_{k-j}, \quad j \in \mathbb{N}, j \underset{c}{<} k.$$

Finally, all the resulting d_k are divided by the independent term of c.

Observe that if the independent term c_0 is 0, then the recursive method can not be applied. Actually if the independent term is zero the division (in an scalar way) is not defined. Therefore it has sense that we cannot compute the division in that case. An exceptional case, in which it can be computed, is when the independent term of b is zero, $b_0 = 0$. Then, if there is a non-zero coefficient of c of multi-index k and its variable is in all the terms of b, we can simplify dividing by that value and operate as usual. For example, if $b(\vec{\xi}) = \xi_1^2 + \xi_1\xi_2$ and $c(\vec{\xi}) = \xi_1 + \xi_1\xi_2 + \xi_1^2$ we can simplify the previous expressions and divide $\xi_1 + \xi_2$ by $1 + \xi_2 + \xi_1$ using the previous formulae.

Power

Given exponent $\alpha \in \mathbb{R}$ and a polynomial $b(\vec{\xi})$, we want to determine the coefficients of $p(\vec{\xi}) = \left(b(\vec{\xi})\right)^{\alpha}$.

We start with the one variable case. If $\alpha \in \mathbb{Z}$ an option is to use the formulae obtained for the product and division. If $|\alpha|$ is small this can be a good option. However, if it is large the number of operations will be cumbersome and the method inefficient. If $\alpha \notin \mathbb{Z}$ $p(\xi)$ must be computed using a different procedure. Taking logarithms in $p(\xi) = (b(\xi))^{\alpha}$, we get:

$$\log(p(\xi)) = \alpha \log(b(\xi)).$$

Then, differentiating with respect the variable ξ we get:

$$\frac{p'(\xi)}{p(\xi)} = \alpha \frac{b'(\xi)}{b(\xi)} \iff p'(\xi)b(\xi) = \alpha p(\xi)b'(\xi).$$

Now, according to Leibniz's formula, and looking at each order, we get:

$$\sum_{i=0}^{n}(i+1)p_{i+1}b_{n-i} = \alpha \sum_{i=0}^{n}(n-i+1)p_i b_{n-i+1}.$$

Observe that each term of this equation is giving the coefficient of order n since we differentiate only once. Now, isolating p_{n+1} in the last formula we get:

$$\begin{aligned}
p_{n+1} &= \frac{1}{(n+1)b_0}\left(-\sum_{i=0}^{n-1}(i+1)p_{i+1}b_{n-i} + \alpha\sum_{i=0}^{n}(n-i+1)p_i b_{n-i+1}\right) \\
&= \frac{1}{(n+1)b_0}\left(-\sum_{i=1}^{n} i p_i b_{n-i+1} + \alpha\sum_{i=0}^{n}(n-i+1)p_i b_{n-i+1}\right) \\
&= \frac{1}{(n+1)b_0}\left(-\sum_{i=0}^{n} i p_i b_{n-i+1} + \alpha\sum_{i=0}^{n}(n-i+1)p_i b_{n-i+1}\right) \\
&= \frac{1}{(n+1)b_0}\left(\sum_{i=0}^{n}\left(-i + \alpha(n-i+1)\right)p_i b_{n-i+1}\right) \\
&= \frac{1}{(n+1)b_0}\left(\sum_{i=0}^{n}\left(\alpha(n+1) - i(1+\alpha)\right)p_i b_{n-i+1}\right).
\end{aligned}$$

Therefore, the final formula for the coefficient of order n is:

$$p_n = \frac{1}{nb_0}\left(\sum_{i=0}^{n-1}(n\alpha - i(\alpha+1))b_{n-i}p_i\right).$$

In the case of multiple variables the same strategy works. Now, since instead of only one variable we have several, to obtain all the coefficients we need to compute all the partial derivatives with respect ξ_i. The equations where we must equate coefficients are:

$$p_{\xi_i}(\vec{\xi})b(\vec{\xi}) = \alpha p(\vec{\xi})b_{\xi_i}(\vec{\xi}), \quad i = 1, \ldots, l.$$

Looking at the terms with multi-index $k - e_i$, where $e_i = (0, \ldots, \overbrace{1}^{i\text{-th}}, \ldots, 0)$, we obtain:

$$\sum_{\substack{j\in\mathbb{N}^l \\ j \leq_c k}} (k-j)]_i p_{k-j}b_j = \alpha \sum_{\substack{j\in\mathbb{N}^l \\ j \leq_c k}} j]_i p_{k-j}b_j, \quad \substack{i=1,\ldots,l \\ k\in\mathbb{Z}^l, |k|\leq n}. \tag{2.5}$$

As in the one-dimensional case, we can isolate the contribution of the degree $|k|$ to p from the remaining terms to get:

$$\sum_{\substack{j\in\mathbb{N}^l \\ j <_c k}} (k-j)]_i p_{k-j}b_j + k]_i p_k b_0 = \alpha \sum_{\substack{j\in\mathbb{N}^l \\ j <_c k}} j]_i p_{k-j}b_j + \underbrace{j]_i}_{0} p_k b_0, \quad \substack{i=1,\ldots,l \\ k\in\mathbb{Z}^l, |k|\leq n}. \tag{2.6}$$

Observe that the term $j]_i$ in the last term is null since it comes from the multi-index $j = 0$. So, isolating the only term of degree $|k|$ of the previous equation we obtain:

$$\begin{aligned} p_k &= \frac{1}{k]_i b_0}\left(\alpha \sum_{\substack{j\in\mathbb{N}^l \\ j <_c k}} (k-j)]_i p_j b_{k-j} - \sum_{\substack{j\in\mathbb{N}^l \\ j <_c k}} j]_i p_j b_{k-j}\right) \\ &= \frac{1}{k]_i b_0}\left(\sum_{\substack{j\in\mathbb{N}^l \\ j <_c k}} \big(\alpha k]_i - j]_i(\alpha+1)\big) p_j b_{k-j}\right) \quad \substack{i=1,\ldots,l \\ k\in\mathbb{Z}^l, |k|\leq n}. \end{aligned}$$

Observe that if $k]_i = 0$ then the previous formula is not defined since both the numerator and the denominator are zero. However, in that case the formulae are useless. If $k = 0$, p_k is computed taking the usual power $p_0 = b_0^\alpha$ as a real number. Otherwise there exist a value i for which $k]_i \neq 0$ and can be used to obtain p_k. This introduces another problem. Using this formulation we have in general l different ways to obtain p_k, one for each derivative that we use. We must be sure that all of them give the same result. Indeed, the term p_k is at the normalised derivative of $b(\vec{\xi})^\alpha$, therefore it is unique and all the possibilities will be equal.

As in the previous cases, we go over the polynomials p and b (starting for the low orders of p) in such way that we compute all the products whose result is of degree $|k|$ and we add them to the proper coefficient.

Sines and Cosines

Let $b(\vec{\xi})$ be a polynomial, and $c(\vec{\xi})$ and $s(\vec{\xi})$ the polynomial expressions for the $\cos(b(\vec{\xi}))$ and $\sin(b(\vec{\xi}))$, respectively. As in the preceding cases, the objective is to find recursive formulae to obtain the values of the coefficients of $c(\vec{\xi})$ and $s(\vec{\xi})$.

In the one variable case we can obtain those formulae by differentiating $c(\xi) = \cos(b(\xi))$ and $s(\xi) = \sin(b(\xi))$, to get:

$$\begin{aligned} c'(\xi) &= -\sin(b(\xi))\, b'(\xi) = -s(\xi)\, b'(\xi), \\ s'(\xi) &= \cos(b(\xi))\, b'(\xi) = c(\xi)\, b'(\xi). \end{aligned}$$

Applying Leibniz's formula, as in the previous cases, and then equating each order n, we get the following formulae:

$$\begin{aligned} c_n &= \frac{-1}{n}\sum_{i=0}^{n} i s_i b_{n-i}, \\ s_n &= \frac{1}{n}\sum_{i=0}^{n} i c_i b_{n-i}. \end{aligned}$$

Observe that the above formulae provide the values of c_n and s_n simultaneously, since the c_n coefficient depends on s_i and vice-versa. Therefore, if we compute one of the two trigonometric functions we get the other one for free.

The procedure is similar in the case of several variables, except that the derivatives must be done with respect all the variables ξ_i. In this way, one gets:

$$\begin{aligned} c_{\xi_i}(\vec{\xi}) &= -s(\vec{\xi})a_{\xi_i}(\vec{\xi}), \\ s_{\xi_i}(\vec{\xi}) &= c(\vec{\xi})a_{\xi_i}(\vec{\xi}). \end{aligned}$$

Equating the terms with multi-index $k - e_i$ we obtain:

$$\begin{array}{rclcrcl} k]_i c_k &=& -\sum_{j\in\mathbb{N}^l, j\leq_c k}(k-j)]_i s_j b_{k-j} & \iff & c_k &=& \frac{-1}{k]_i}\sum_{j\in\mathbb{N}^l, j<_c k}(k-j)]_i s_j b_{k-j} \\ k]_i s_k &=& \sum_{j\in\mathbb{N}^l, j\leq_c k}(k-j)]_i c_j b_{k-j} & & s_k &=& \frac{1}{k]_i}\sum_{j\in\mathbb{N}^l, j<_c k}(k-j)]_i c_j b_{k-j}. \end{array}$$

Observe that when $j = k$ the term $(k-j)]_i$ banishes, therefore we can remove all the terms of order $|k|$. As in the power formula, in general we have l different formulae to compute c_k and s_k. The values of c_k and s_k give the normalised derivative of the sines and cosines, which is unique, therefore, all the formulae give the same result.

As in the previous two cases, instead of looking for all the terms with multi-index k, in the implementation we do the product of those terms of order $|k|$ and then we add them to the corresponding coefficient.

Exponential

Let $a(\vec{\xi}) = e^{b(\vec{\xi})}$. If the polynomial is in one variable, we can easily obtain the coefficients of $a(\xi)$ taking logarithms and differentiating:

$$\log(a(\xi)) = b(\xi) \implies \frac{a'(\xi)}{a(\xi)} = b'(\xi) \iff a'(\xi) = b'(\xi)a(\xi).$$

Applying, once again, Leibniz's formula and equating the terms of order $n-1$ in ξ, we get:

$$n a_n = \sum_{i=0}^{n}(n-i)b_{n-i}a_i = \sum_{i=0}^{n-1}(n-i)b_{n-i}a_i \iff a_n = \frac{1}{n}\sum_{i=0}^{n-1}(n-i)b_{n-i}a_i.$$

For several variables we proceed in an analogous way to obtain:

$$a_{\xi_i}(\vec{\xi}) = b_{\xi_i}(\vec{\xi})a(\vec{\xi}).$$

Comparing those terms with multi-index $k - e_i$, the resulting formula is:

$$a_k = \frac{1}{k]_i}\sum_{\substack{j\in\mathbb{N}^l \\ j<_c k}}(k-j)]_i b_{k-j}a_j.$$

Again, as in the other cases, we obtain l different formulae to compute the same argument; since a_k is the k-th normalised derivative, all of them give the same result. As before, when implementing the formulae we go over the polynomials a and b to obtain the coefficients of order $|k|$, then each product is assigned to the corresponding coefficient instead of finding the products that give exactly the multi-index k.

Logarithm

Given a polynomial $b(\vec{\xi})$ the objective is to compute the polynomial $a(\vec{\xi}) = \log(b(\vec{\xi}))$. In the case of polynomials in one variable the recursion can be determined by differentiation with respect to ξ, to get:

$$a'(\xi) = \frac{b'(\xi)}{b(\xi)} \iff a'(\xi)b(\xi) = b'(\xi).$$

Then, applying Leibniz's formula and equating the terms of order $n-1$ we get:

$$a_n = \frac{1}{b_0}\left(b_n - \frac{1}{n}\left(\sum_{i=1}^{n-1} i a_i b_{n-i}\right)\right).$$

To obtain the formulae for several variables we proceed in analogous way, but now differentiating with respect the variables ξ_i:

$$a_{\xi_i}(\vec{\xi})b(\vec{\xi}) = b_{\xi_i}(\vec{\xi}).$$

After equating the terms of order $k - e_i$ we obtain:

$$k]_i b_k = \sum_{\substack{j\in\mathbb{N}^l \\ j <_c k}} j]_i a_j b_{k-j} + k]_i a_k b_0.$$

Finally, isolating the unique term of order $|k|$ in this equation the final formula is obtained:

$$a_k = \frac{1}{b_0}\left(b_k - \frac{1}{k]_i}\left(\sum_{\substack{j\in\mathbb{N}^l \\ 0<|j|<|k|}} j]_i a_j b_{k-j}\right)\right).$$

As happens when we use the differentiation to deduce the formulae, we obtain l different formulae to compute the same coefficient. Since the coefficients of a are the normalised derivatives of the logarithm, there is a unique value for them, therefore, the l formulae must coincide.

2.2.2 Testing the polynomial algebra

In the present section we present the results of some of the tests and comparisons that have been done to evaluate the accuracy and efficiency of the implemented polynomial algebra package.

To perform the we have used Pari/GP [The14]. This package implements two relevant things: from one side it supports multi-precision arithmetic. This is very suitable to check the accuracy of our computations, since the solution provided with Pari/GP has very low truncation errors and can be considered a reference. From another side, Pari/GP also includes polynomial algebra computations making some of the tests even simpler.

As a first test, we have performed the basic algebraic operations using our own algebra package and Pari/GP. Subtracting the resulting polynomials we can see how much the coefficients differ. These tests have been done, for each algebraic operation, 1000 times (using randomly generated polynomials) in order to compute the averaged errors. As we have already mentioned, the accuracy of Pari/GP is higher than the machine precision, therefore the computations done with Pari/GP can be used to validate the ones of the algebra package. The coefficients of the initial polynomial and the scalars of the different operations are selected using uniform distribution in the following intervals:

- Division $d(\vec{\xi}) = b(\vec{\xi})/c(\vec{\xi})$.
  The coefficients of b are selected in the interval $(-10, 10)$.
  The coefficients of c are selected in the interval $(0, 20)$.

- Power $p(\vec{\xi}) = b(\vec{\xi})^\alpha$:
  The coefficients of b are selected in the interval $(0, 20)$.
  The alpha is selected in the interval $(-2.5, 2.5)$.

- Sines and cosines ($c(\vec{\xi}) = \cos(b(\vec{\xi})), s(\vec{\xi}) = \sin(b(\vec{\xi}))$:
  The coefficients of b are selected in the interval $(-10, 10)$.

- Exponential ($a(\vec{\xi}) = e^{b(\vec{\xi})}$:
  The coefficients of b are selected in the interval $(-10, 10)$.

- Logarithm $a(\vec{\xi}) = log(b(\vec{\xi}))$:
  The coefficients of b are selected in the interval $(0, 20)$.

Table 2.1 shows the decimal logarithm of the error for the power ($b^\alpha(\vec{\xi})$), the division ($b(\vec{\xi})/c(\vec{\xi})$), and the trigonometric functions ($\sin(b(\vec{\xi}))$, $\cos(b(\vec{\xi}))$), taking polynomials $b(\vec{\xi})$ and $c(\vec{\xi})$ with two variables and degree three. It can be seen that the precision of the computations is close to the machine precision.

k_0	k_1	power	division	sines	cosines
0	0	-14.41550	-15.8465	-16.7759	-16.495
1	0	-14.98994	-15.5148	-15.8057	-15.8871
0	1	-15.05401	-15.5068	-15.7906	-15.8517
2	0	-14.77298	-15.1239	-15.137	-15.1591
1	1	-14.56881	-14.9689	-14.9576	-14.9978
0	2	-14.83939	-15.1173	-15.1112	-15.1446
3	0	-14.61939	-14.7187	-14.5815	-14.5823
2	1	-14.40447	-14.356	-14.3343	-14.3222
1	2	-14.38862	-14.2958	-14.3417	-14.3127
0	3	-14.66474	-14.6576	-14.5848	-14.563

Table 2.1: Decimal logarithm of the differences between the coefficient associated to the exponents k_0, k_1, computed using the polynomial algebra and Pari/GP for some basic functions: power, quotients, sines and cosines. For the computations with Pari/GP multi-precision arithmetic has been used.

The same kind of computations can be done for the logarithm and the exponential of polynomials. Table 2.2 shows the errors obtained for these two functions. Due to some problems with PARI in the computation of the logarithm, for this function first we check that the exponential is correctly computed with PARI, and then we use the exponential to check that the logarithm computed with our software is correct doing the composition of both operations. Nevertheless, the logarithm computation has bigger errors since it accumulates the errors of both operations.

Table 2.3 gives the CPU time used with an Intel Xeon(R), CPU E5645, 2.40GHz, for some of the basic functions. It must be noted that, in Pari/GP, the quotient of two polynomials is, in general, a rational function and not its Euclidean quotient. Also, the computation of sines and cosines using Pari/GP must be done with two independent functions and with our code both trigonometric functions are computed simultaneously.

An important point when we work with jets is which is the order used. For instance, if we are only interested in the first order expansion, this can be the case when we only want to solve the first variational equations (see section 2.3 for further reference), we only need jets of order one. In those cases it is possible to use a specific storage system. Doing so the time consumption is drastically reduced. In section 2.3 some results to compare the two implementations are presented.

As it has already been mentioned, another point that must be taken into account when implementing the Jet Transport is the storage system. Some tests to compare our storage system with other implementations has been done. In particular the storage system presented here has been compared with the nested trees

k_0	k_1	exponential	logarithm
0	0	-17.5671	-14.9621
1	0	-18.1354	-14.9856
0	1	-18.1819	-15.0018
2	0	-16.6722	-14.7911
1	1	-17.0234	-14.7211
0	2	-16.6189	-14.8179
3	0	-15.2986	-14.3592
2	1	-15.8441	-14.2395
1	2	-15.8086	-14.2419
0	3	-15.2924	-14.3999

Table 2.2: Decimal logarithm of the difference of the coefficient associated to the exponents k_0, k_1, computed using the polynomial algebra and Pari/GP for the exponential (third column) and the logarithm. See more explanations in the text.

	Code	Pari/GP
product	18.669s	3m 26.225s
quotient	35.358s	10.687s
sin/cos	1m 38.014s	6m 55.458s
power	3m 5.766s	117m 40.329s

Table 2.3: CPU time for 10^6 iterations of some basic functions with polynomials in 2 variables and degree 8 using our code and PariGP (Intel Xeon(R), CPU E5645, 2.40GHz). For the quotient see the text for additional notes.

method implemented by Zubin Olikara [Oli15], based in Chapter 2 of the forthcoming book [HCF$^+$15]. This method is faster than ours. The main reason for this is that in it the memory access does not require as many operations as our procedure. Figure 2.1 shows the CPU times spent by the two methods to compute 1000 times the product of two multivariate polynomials of the given order and variables. It also gives the ratio between both. We can see that for low orders and a small number of variables the nested tree is five times faster. However, for larger orders and number of variables it is up to 20 times faster than the method implemented here.

Therefore, a foreseen improvement of the package is the implementation of some of the basic core operations to handle the nested tree storage method.

2.2.3 Differential polynomial algebra

As we already stated, the objects in the polynomial algebra are truncated polynomials $P(\vec{\xi})$. Each polynomial in the algebra is the local representative of a set of functions: the polynomial itself and all the functions f such that $P(\vec{\xi})$ coincides with the first terms of its Taylor expansion.

In what follows, we will give the procedures for the computation of the derivatives and integrals of polynomials.

Given a function f it is possible to differentiate and integrate with respect to one of its variables. Therefore, it has sense to consider those operations also in the polynomial algebra.

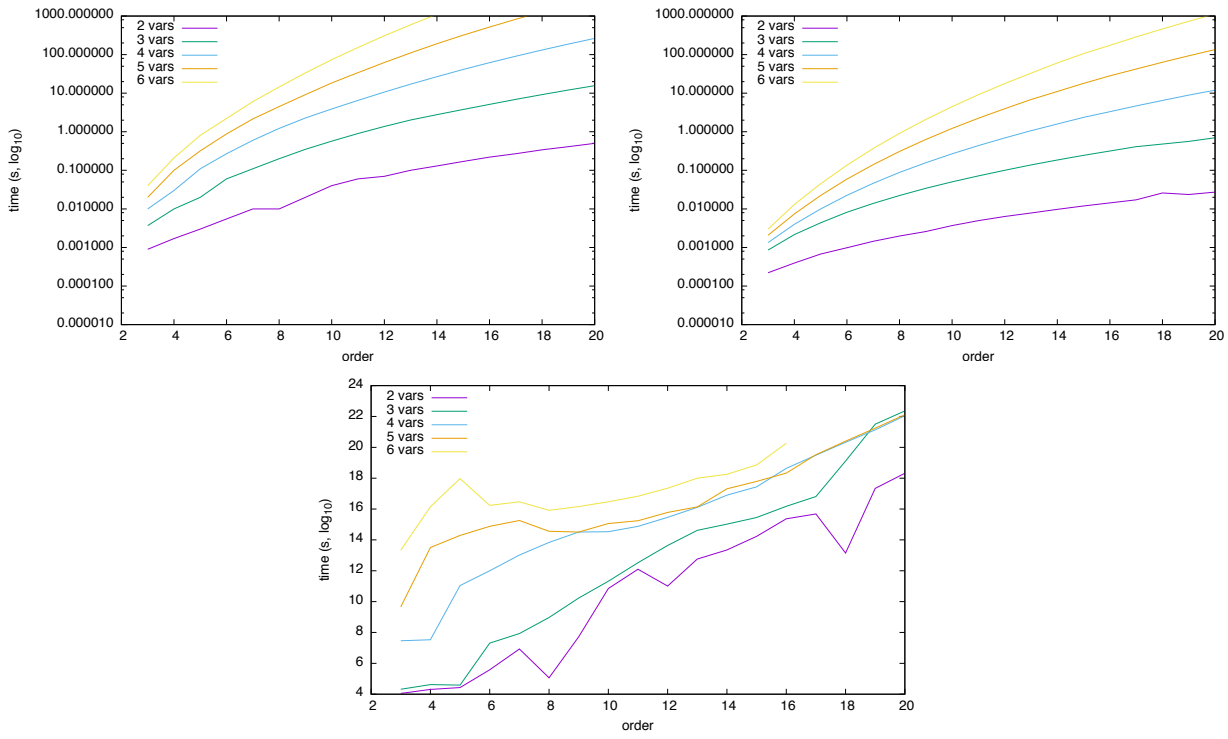


Figure 2.1: Comparison between the algebra presented and the nested tree algebra. Top left: CPU time spent using the implemented polynomial algebra to compute 1000 products of two random polynomials. Top right: CPU time spent using the nested tree storage polynomial algebra to compute 1000 products of the same random polynomials. Bottom: Ratio between the implemented method and the nested tree storage system. In all the cases the colour gives the number of variables used and the x-axis variable the order used.

Differentiation of polynomials

Given a multivariate polynomial $P(\vec{\xi}) = \sum_{\substack{k\in\mathbb{N}^l \\ |k|\leq n}} a_k \vec{\xi}^k$ we can compute its derivative with respect to the variable ξ_i with the usual differentiation laws:

$$P_{\xi_i}(\vec{\xi}) = \sum_{\substack{k\in\mathbb{N}^l \\ |k|\leq n}} k]_i a_k \vec{\xi}^{k-e_i}.$$

Each time that we differentiate a polynomial of the truncated polynomial algebra, we loose one order. This happens because if $P(\vec{\xi})$ has degree n, when we differentiate the terms of order n they become of order $n-1$. Therefore, the differentiation of polynomials can be done, but each time it is performed there is some information that is lost.

As we will see in later sections, the differentiation of polynomials is useful to determine zeros, maxima and minima of functions, and it is possible to implement automatic Newton procedures to obtain the desired solutions. The only thing that is needed is to approximate the function, using the corresponding polynomial operations, then compute its derivative, using the formulas presented here, and finally dividing the function by its derivative to get the polynomial with which one can compute the next iterate of Newton's method. This idea can also be translated to higher dimensions for multivariate functions, but some polynomial linear algebra is required; it will be presented in the next section.

Integration of polynomials

The second differential operation is integration. Under the same situation of the differentiation, given a multivariate polynomial $P(\vec{\xi}) = \sum_{\substack{k\in\mathbb{N}^l \\ |k|\leq n}} a_k \vec{\xi}^k$ we can compute its indefinite integral with respect to ξ_i using the integration law for polynomials.

$$\int P(\vec{\xi}) d\xi_i = \sum_{\substack{k\in\mathbb{N}^l \\ |k|\leq n-1}} \frac{1}{k]_i + 1} a_k \vec{\xi}^{k+e_i}.$$

Observe that, as it happens with the differentiation, when doing the integration we also lose some information. The coefficients that initially had order n, after the integration will have order $n+1$. However, since we are working in a truncated polynomial algebra, all the polynomials obtained must be of order n, therefore, those terms of order $n+1$ are neglected in by the process. This justifies the addition of the $n-1$ as maximum order in the formula. Using the previous formula, the integrated polynomial does not have independent term. With the usual integration procedure, the independent term is a constant, c, that must be determined. Observe that when the polynomial is multivariate, the independent term when we integrate with respect a certain component on ξ depends on the remaining components, so more information is required for its computation.

If the final objective is to compute a definite integral, then we get:

$$\int_{x_0}^{x_1} P(\vec{\xi}) d\xi_i = \sum_{\substack{k\in\mathbb{N}^l \\ |k|\leq n-1}} \frac{1}{k]_i + 1} a_k (x_1^{k]_i+1} - x_0^{k]_i+1}) \vec{\xi}^{k-k]_i e_i},$$

where the super-index $k]_i$ denotes the i-th component of the array k.

2.2.4 Polynomial linear algebra

We have already mentioned the possibility of using Newton's method with polynomials in several variables; in this situation, to generate the iterative procedure defined by the method, we need to operate with matrices and vectors of polynomials. The polynomial linear algebra functions provide some additional functionalities to the polynomial algebra. We have already mentioned, as a first example, an automatic Newton solver of non-linear equations, another example is the polynomial expansion of the inverse of a function, computed from the polynomial expansion of the function, that will be introduced in Section 2.6.

One can operate with matrices and vectors of polynomials as if they were of real numbers, using for all the elementary operations those implemented in the polynomial algebra. In particular, we can implement any linear algebra algorithm, for instance an $A = LU$ matrix decomposition. The final result will be the coefficients of the two polynomial matrices, L and U, as a function of those of the initial matrix A. If we evaluate all the polynomials at $\vec{\xi} = 0$, this is, we only consider the independent term of the polynomials, what is expected to get is the usual LU factorisation of A. Of course, some special care must be taken in the selection of pivots since, in principle, it is not clear what maximum means with polynomials. One possibility is to consider some Sovolev norm to compute it, however, and in order to respect the premise that the order-zero results must agree with the ones obtained with the standard numerical procedure, it is convenient to define the maximum according to the value of the independent terms of the polynomials involved. Once the LU decomposition has been obtained, it is possible to compute the determinant of the matrix, or the solution of a linear system of equations.

One must always have in mind that all the resulting polynomials will give a good approximation only around the point about which the initial polynomials have been determined. The further we move the worst the results will be

2.2.5 Additional tools of the polynomial algebra

To end this section polynomial algebra section, we will introduce some tools that, although they do not operate with polynomials, they give information about them. The first one is useful to have, in some cases,

an estimation of the radius of the neighbourhood around an initial point $\vec{x}_0$ that we can propagate, this is, how far from $\vec{x}_0$ we can go in the evaluation of a resulting polynomial $P(\vec{\xi})$. The other tool is the evaluation of polynomials; in one variable polynomials it is know Horner's evaluation in order to save computation time, here a generalisation of Horner's rule is presented for more variables.

Maximum size of precision

In order to estimate the size of a neighbourhood U that we can propagate, within a given precision, using the polynomial algebra, a method inspired in the automatic decision of steps in ordinary differential equations solvers is used. Analogously to what happens in ODE's solvers with step-size control, there is also a domain (box) where the polynomial obtained using the polynomial algebra has errors below a given tolerance. The problem is how to estimate the size of this box. In analogy with the step-size control selection, some estimations can be done using the higher order terms on the final jet.

Let

$$P(\vec{\xi}) = \sum_{|k|\leq n} a_k \vec{\xi}^k,$$

be the polynomial obtained at the end of the procedure, with $\vec{\xi} = (\xi_1, \xi_2, ..., \xi_l)$ and where $a_k = (a_{1,k}, \ldots, a_{l,k})$, and $a_{i,k}$ is the coefficient of the i-th variable with multi-index exponent k. A straightforward option is to select the size of the box in such way that the contribution of last term to P is below a certain tolerance ε_{jet}, i.e: $a_{m,k}\xi^k \leq \varepsilon_{\text{jet}}$.

Assuming that the box has to be of the same size in any direction, we get that this size is:

$$\xi_{\max} = \min_{\substack{|k|=n \\ 1\leq m\leq l}} \left(\frac{\varepsilon_{\text{jet}}}{a_{m,k}}\right)^{1/k}. \tag{2.7}$$

This selection procedure must be modified when the last term of the final (propagated) polynomial is zero, since in this case it will not be true that, in general, $\xi_{\max}$ can be as large as we want. Consider, for instance, the series of the scalar function $f(x) = \sin(x)$, around $x = 0$, with a polynomial algebra truncated at order 4. In this case, the jet of f is:

$$P(\xi) = \xi + 0\,\xi^2 - \frac{1}{6}\,\xi^3 + 0\,\xi^4.$$

Clearly, the above formula for $\xi_{\max}$ gives an infinite radius, but just looking to the Taylor polynomial of F truncated at order 5 one sees that this term is positive, so there will be some error.

In these cases an alternative must be considered. For instance, one can study the decay of the coefficients of the polynomial; if they do it in a proper and expected way, then the series will converge in a wider range, instead, if they do not decay or they do not decay fast enough, then the series will have smaller radius of good approximation (see [WDLA$^+$15]).

Unfortunately, there is not a general analytical procedure to do a priori estimations of the error.

Evaluation of polynomials

Let P_f be a polynomial approximation of a certain function f. If we apply P_f to the polynomial $\vec{x}_0 + \vec{\xi}$, we get a new polynomial P. For each value of $\vec{\xi}$, $P(\vec{\xi})$ will be an approximation of $f(\vec{x}_0 + \vec{\xi})$. The smaller the value of $\vec{\xi}$ is, the better the approximation will be. If $\vec{\xi} = 0$ the error will be only due to the round off errors. Therefore, the polynomial algebra can be applied to compute, within a certain accuracy, the values of f in a certain neighbourhood U of $\vec{x}_0$, parametrised by the vector $\vec{\xi}$. The computation will be fast if the evaluation of P is fast.

A case of interest, in which a great amount function evaluations are required, are Monte Carlo simulations. A Monte Carlo simulation of a certain system, modelled by some function or a numerical procedure, consists in the evaluation of the function or the execution of the numerical procedure several hundreds or thousands

of times, each time with different realisations of the random variables of the system. Then, the resulting samples are used to extract statistical information of the system. If the modelling function, or the output of the numerical procedure, can be approximated by polynomials, the implementation of the Monte Carlo can be done very fast, compared with the standard procedure, if the evaluation of the polynomials is fast.

Therefore, a good polynomial evaluation routine is an important ingredient in any polynomial algebra implementation.

For polynomials in one variable, the well known Horner's algorithm is the best option for its evaluation, since it involves the minimum number of operations. Given $P(\xi) = \sum_{i=0}^{n} a_i \xi^n$, the method consist in extract as many ξ as possible as common factor, in such way that the polynomial is written as $P(\xi) = a_0 + \xi(a_1 + \xi(\cdots + \xi(a_{n-1} + a_n\xi)))$. Then, the procedure consists in doing the computations from the inner most parenthesis to the outermost one.

In the multivariate case, we can also apply a modification of Horner's method. Remember that a polynomial $P_{l,n}(\vec{\xi})$ with l variables and order n can be seen as a polynomial $P_n(\xi_1)$ whose coefficients are polynomials of $l-1$ variables and order, at most, n:

$$P_{l,n}(\xi) = P_n(\xi_1) = \sum_{i=0}^{n} P_{l-1,n-i}(\bar{\xi})\xi_1^i.$$

Using this fact, we can use Horner's method recursively to evaluate the polynomial. In a first step, we apply Horner's method to the above polynomial in the variable ξ_1. To evaluate the products of the polynomial coefficients by the powers of ξ_1, we use again Horner's method to evaluate a polynomial with one variable less and order equal to the one of the original polynomial minus the power of the variable of the current step. The recursion is repeated until there is only one remaining variable, then the usual Horner's method is applied.

Observe that the reverse lexicographic order is well suited to the evaluation of polynomials. With this ordering, as it was already explained, the coefficients are ordered by variables and degrees. Therefore, the first block of coefficients contains only the terms which do not depend on ξ_l, the second block contains the coefficients that depend on ξ_l linearly, after come the blocks that depend quadratically, and so on until the last coefficient contains the coefficient of $\vec{\xi}_l^k$. Therefore, in order to apply the multivariate Horner's evaluation it is interesting to start not by the first variable but by the last one. Doing so the polynomial is already ordered.

If instead of evaluate the polynomial at real numbers we evaluate it at polynomials, this is, if the variable $\vec{\xi}$ is $Q_i(\bar{\xi})$, then the above algorithm provides a way to do the composition of two polynomials that, in principle, should be avoided as much as it is possible. It must also be said that this way of doing the composition avoids most of the power computations.

A particular case of evaluation appears in the computation of definite integrals. We have already seen that given a polynomial $P(\vec{\xi})$ we can compute the definite integral using:

$$\int_{x_0}^{x_1} P(\vec{\xi}) d\xi_i = \sum_{\substack{k \in \mathbb{N}^l \\ |k| \leq n-1}} \frac{x_1^{k]_i+1} - x_0^{k]_i+1}}{k]_i + 1} a_k \vec{\xi}^{k-k]_i e_i}.$$

Observe that the right hand of the equation is just the 1-dimensional evaluation of the polynomial P. Therefore, to compute a definite integral we can evaluate, using the classical Horner's method, the polynomial obtained doing the indefinite integral twice, one evaluating x_1 and the other evaluating x_0, and subtracting them.

2.3 Flow propagation

The flow propagation of jets is one of the main tools that has been developed. This tool has already introduced in [AFV$^+$09, AFJ$^+$08] and is also known as Jet Transport.

Recall that the jet of a function is the truncated Taylor polynomial and what the tool does is to propagate the jet of a given function through the flow of a vector field, this is, to transport the jet from one initial state to the final.

The Jet Transport is a tool that allows the computation of the image, under the flow map, of a set of points around an initial state $\vec{x}_0$. In contrast, the usual numerical integration methods give only the image of the state $\vec{x}_0$. To obtain this map, Jet Transport techniques use any numerical integration method for ordinary differential equations (ODE) to obtain the solutions of the variational equations associated to the ODE up to a high order, without writing them explicitly. The direct computation of high order variational equations requires a large amount of work just to obtain the system of differential equations that must be integrated, this is avoided when the Jet Transport is used.

To describe the methodology, consider the system $\dot{\vec{x}} = f(t, \vec{x})$ and the associated flow map $\phi(t; t_0, \vec{x}_0) = \vec{x}_t$ which gives the position at time t of a particle that at time t_0 is at $\vec{x}_0$. The objective is to propagate a full neighbourhood U of $\vec{x}_0$, i.e. to compute $\phi(t; t_0, U)$. For this purpose, the initial condition $\vec{x}_0$ is replaced by a polynomial that parametrises its neighbourhood: $P_{t_0,\vec{x}_0}(\vec{\xi}) = \vec{x}_0 + \vec{\xi} = \vec{x}_0 + (\xi_1, \ldots, \xi_n)^T$, where $\vec{\xi}$ represents the deviation with respect to $\vec{x}_0$.

Now, any ordinary differential equation integrator (Runge-Kutta, Taylor, symplectic,...) can be used to propagate $\vec{x}_0 + \vec{\xi}$. Note that, since the initial condition is a polynomial, all the computations required by the integration method have to be done using the polynomial algebra already introduced. In this way we obtain $P_{t,\vec{x}_0}(\vec{\xi})$, the expression of the neighbourhood U propagated up to time t by means of truncated Taylor's series up to order n. These series give the image under the flow at time t of the initial states $\vec{x}_0 + \vec{\xi}$, i.e. $P_{t,\vec{x}_0}(\vec{\xi}) = \phi(t; t_0, \vec{x}_0 + \vec{\xi})$.

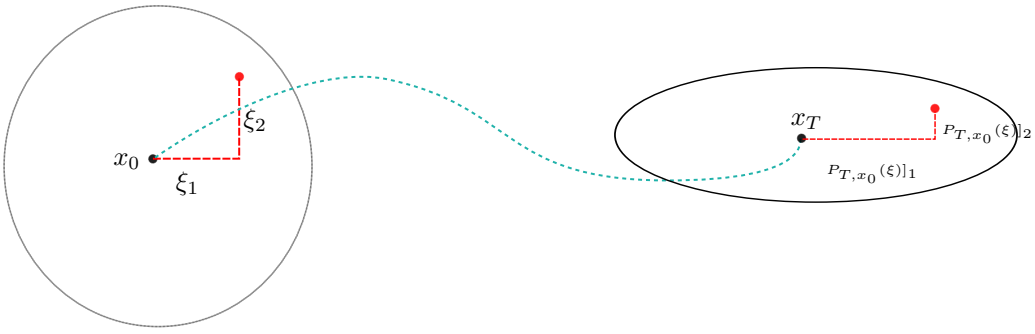


Figure 2.2: Schema of the basic idea underlying the Jet Transport.

We will start this section giving a first example of how the Jet Transport works. We will see that the main tools will be:

- the polynomial algebra,
- and any differential equation integrator adapted to hold jet operations.

Later, we will recall a well known ODE integrator method, the Taylor method for the numerical integration of ordinary differential equations. It is, in fact, a method that uses intensively the techniques of the polynomial algebra. We will also talk about how to adapt the Taylor method to handle polynomials and how the step size can be selected. Finally, we will end the section presenting the numerical results obtained using some test examples: a linear saddle, a standard and a perturbed pendulum and the CR3BP.

2.3.1 A first example

As an illustration of how the Jet Transport works, consider a simple example using a linear centre $f(\vec{x}) = (y, -x)$ and the simplest integration method: the well known Euler's method. According to this method, at

each iteration step the system evolves as:

$$\vec{x}_{n+1} = \vec{x}_n + h\, f(\vec{x}_n).$$

Now, instead of using the initial condition $\vec{x}_0 = (x_0, y_0)$, we take the polynomial,

$$P_{0,\vec{x}_0}(\xi_1, \xi_2) = (x_0 + \xi_1, y_0 + \xi_2),$$

which represents the initial condition plus some perturbation in its neighbourhood.

After a first step of Euler's method, recalling that all the computations must be done with polynomials instead of real numbers, we get:

$$\vec{x}_1 = P_{h,\vec{x}_0}(\vec{\xi}) \quad = \quad P_{0,\vec{x}_0}(\vec{\xi}) + h\, f(P_{0,\vec{x}_0}(\vec{\xi})) = \begin{pmatrix} x_0 + h\, y_0 \\ y_0 - h\, x_0 \end{pmatrix} + \underbrace{\begin{pmatrix} 1 & h \\ -h & 1 \end{pmatrix}}_{M(t_0+h)} \begin{pmatrix} \xi_1 \\ \xi_2 \end{pmatrix}.$$

Note that the independent term of the polynomial $P_{h,\vec{x}_0}(\vec{\xi})$, with respect to the variable $\vec{\xi}$, corresponds to a first step of the usual Euler's method without using the Jet. On the other hand, the matrix $M(t_0 + h)$ contains the solution, using Euler's method, of the first variational equations. We also remark that there are no higher order terms, because the solution of the second order variational equations is 0. So, what we have obtained is, $\phi(0; h, \vec{x}_0 + \vec{\xi})$.

We can iterate again the procedure to obtain the results for a second Euler's step:

$$P_{2h,\vec{x}_0}(\vec{\xi}) = \begin{pmatrix} x_0 + 2hy_0 - h^2 x_0 \\ y_0 - 2hx_0 - h^2 y_0 \end{pmatrix} + \begin{pmatrix} 1 - h^2 & 2h \\ 2h & 1 - h^2 \end{pmatrix} \begin{pmatrix} \xi_1 \\ \xi_2 \end{pmatrix}.$$

Again, the two terms obtained correspond to the image of the initial condition x_0 after this second step (the independent term), and the solution of the first order variational equations for the degree one terms.

Although in the above example we have used Euler's method, any other numerical integration algorithm would work. For instance, using a Runge-Kutta-4 the results are:

$$P_{h,x_0}(\vec{\xi}) = \begin{pmatrix} 1 - \frac{1}{3}h^2 + \frac{1}{24}h^4 & \frac{2}{3}h - \frac{1}{12}h^3 \\ -\frac{2}{3}h + \frac{1}{12}h^3 & 1 - \frac{1}{3}h^2 + \frac{1}{24}h^4 \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + \underbrace{\begin{pmatrix} 1 - \frac{1}{3}h^2 + \frac{1}{24}h^4 & \frac{2}{3}h - \frac{1}{12}h^3 \\ -\frac{2}{3}h + \frac{1}{12}h^3 & 1 - \frac{1}{3}h^2 + \frac{1}{24}h^4 \end{pmatrix}}_{M(t_0+h)} \begin{pmatrix} \xi_1 \\ \xi_2 \end{pmatrix}.$$

Therefore, the only tools needed for the Jet Transport method are: a numerical integration procedure and a polynomial algebra package. We already introduced the latter one. Let us introduce now how to adapt a numerical integration procedure.

2.3.2 Taylor's method for ODEs

There are many integration methods to compute the solution of an ordinary differential equation. Each one of them with their own weaknesses and strengths. We have just shown how to use the jet with an Euler method and gave some results using a RK-4. However, any method can be used. The method that has been used in this work is Taylor's method. This method uses the automatic differentiation procedures, provided by the polynomial algebra computations, to get high order derivatives of the flow map. It gives very good results for non-stiff differential equations.

Consider the polynomial expansion of the flow with respect to the independent variable:

$$\phi(t_0 + h; t_0, \vec{x}_0) = \vec{x}(t_0 + h; t_0, \vec{x}_0) = \sum_i x^{(i)} h^i = \phi(t_0; t_0, \vec{x}_0) + \frac{\partial \phi(t_0; t_0 \vec{x}_0)}{\partial t} h + \frac{1}{2} \frac{\partial^2 \phi(t_0; t_0 \vec{x}_0)}{\partial t^2} h^2 + \dots .$$

In order to approximate the value of $\partial\phi(t_0;t_0\vec{x}_0)/\partial t$ we can use the fact that the vector field $f(\vec{x},t)$ is tangent to the flow, therefore:

$$\frac{\partial\phi(t_0;t_0\vec{x}_0)}{\partial t} = f(\vec{x},t).$$

Euler's method uses only the constant and linear terms of the expansion of the flow; if we want better approximations then, according to the above equation, more derivatives of the vector field will be needed. Several methods, like those of the Runge-Kutta family, use closed formulae for the computation of the coefficients of the expansion of ϕ. Taylor's method avoids closed formulae using automatic differentiation for the computation of the derivatives, according to the formulae of the previous section. Once obtained, they are normalised and inserted in the flow expansion.

The estimation of the step size h in Taylor's method is done in such way that the truncated series, approximating the solution, have an error smaller than a certain tolerance ε. The selection of h is done looking at the remainders of the polynomials (more concretely, to the last two coefficients of the series), according to the following analysis [JZ05, Sim01].

Assume that the vector field is analytic, therefore, the flow map will be also analytic, hence, the derivatives of the flow are bounded by:

$$\frac{c_1}{\rho^i} \leq |x_k^{(i)}| \leq \frac{c_2}{\rho^i},$$

where c_1 and c_2 are constants verifying $0 < c_1 < c_2$, $c_2 < \gamma c_1$, and ρ is the radius of convergence of the series. The final error committed has to be smaller than a given quantity ε. Let us bound the reminder R_N when the series expansion of ϕ is truncated at order N. We can write

$$\begin{aligned} |(R_N)_k| &= \left|\sum_{j=N+1}^{\infty} x_k^{(j)} h^j\right| \leq \sum_{j=N+1}^{\infty} \frac{c_2}{\rho^j} h^j \\ &= c_2 \sum_{j=N+1}^{\infty} \left(\frac{h}{\rho}\right)^j = \frac{c_2 \left(\frac{h}{\rho}\right)^{N+1}}{1-\frac{h}{\rho}}, \end{aligned}$$

where we have assumed that $h < \rho$, this is, that the step size is smaller than the radius of convergence of the series. If this condition is not verified then we will evaluate the series outside its radius of convergence, which has no sense.

Applying the lower bound for $i = N$, $c_1 \leq \rho^N |x_k^{(N)}|$, we get:

$$\begin{aligned} |(R_N)_k| &\leq \frac{c_2 \left(\frac{h}{\rho}\right)^{N+1}}{1-\frac{h}{\rho}} \leq \frac{c_1\gamma \left(\frac{h}{\rho}\right)^{N+1}}{1-\frac{h}{\rho}} \\ &\leq \frac{\rho^N \gamma |x_k^{(N)}| \left(\frac{h}{\rho}\right)^{N+1}}{1-\frac{h}{\rho}} = \frac{\gamma\frac{h}{\rho}}{1-\frac{h}{\rho}} |x_k^{(N)}| h^N. \end{aligned}$$

Assuming that $\frac{h}{\rho} \leq \frac{1}{2}$:

$$|(R_N)_k| \leq \gamma |x_k^{(N)}| h^N,$$

so, if we ask the reminder to be bounded by ε, the condition that we obtain is:

$$\gamma |x_k^{(N)}| h^N \leq \varepsilon \implies h \leq \left(\frac{\varepsilon}{\gamma |x_k^{(N)}|}\right)^{\frac{1}{N}}. \tag{2.8}$$

This gives a bound for the step size for each variable. Using the maximum norm we obtain a bound for all the variables at once:

$$h \leq \left(\frac{\varepsilon}{\gamma \|x^{(N)}\|_\infty}\right)^{\frac{1}{N}}.$$

It may happen that the series has $x^{(N)} = 0$, for instance if the flow is an odd or even function. To take into account these cases, it is interesting to consider the two last terms of the series instead of only the last one:

$$h_{\text{opt}} = \min\left\{ \left(\frac{\varepsilon}{\gamma \|x^{(N)}\|_\infty} \right)^{\frac{1}{N}}, \left(\frac{\varepsilon}{\gamma \|x^{(N-1)}\|_\infty} \right)^{\frac{1}{N-1}} \right\}.$$

According to the following Proposition (Simó [Sim01]), the most efficient time step is given by $h = e^{-2}\rho$:

Proposition 1. *If the coefficients of the series $x(t_0 + h) = \sum_{i=0}^{\infty} x^{(i)} h^i$ verify the condition*

$$\frac{c_1}{\rho^k} < \left| x^{(k)} \right| < \frac{c_2}{\rho^k},$$

and the number of operations by step are of order N^2, then the most efficient step size tends to $h = \frac{\rho}{e^2}$ when ε tends to 0.

Proof. Define $\widehat{h} = \frac{h}{\rho}$ and $\widehat{\varepsilon} = \frac{\varepsilon}{\gamma}$. Using the lower bound and the bound obtained in (2.8) we obtain:

$$\frac{c_1}{\rho^N} \leq \left| x^{(N)} \right| \leq \frac{\widehat{\varepsilon}}{h^N},$$

therefore

$$\widehat{\varepsilon} \geq \frac{c_1 h^N}{\rho^N} = c_1 \widehat{h}^N,$$

and taking the maximum value for $\widehat{h}^N$:

$$N = \frac{\log \widehat{\varepsilon}}{\log \widehat{h}}.$$

On the other hand, if the total time of integration is l then:

- The number of steps is $\frac{l}{h}$.
- The cost for each step is assumed to be AN^2.

Then, the total cost of the integration is:

$$C(h) = \frac{l}{h} A N^2 = \frac{B N^2}{h},$$

with $B = lA$. So we want to minimise

$$\min C = \frac{B N^2}{h}, \quad \text{such that} \quad N = \frac{\log \widehat{\varepsilon}}{\log \widehat{h}}.$$

Using the constraint and taking into account that $\frac{\rho}{(\log \varepsilon)^2 B}$ is constant, to minimise C is equivalent to minimise $\tilde{C}$:

$$\min_{\widehat{h}} C(\widehat{h}) = \min_{widehath} \frac{B}{h} \left(\frac{\log \varepsilon}{\log \widehat{h}} \right)^2 = \min_{\widehat{h}} \tilde{C}(\widehat{h}) = \frac{\rho}{(\log \varepsilon)^2 B} C(\widehat{h}) = \min_{\widehat{h}} \frac{\rho}{h} \left(\frac{1}{\log \widehat{h}} \right)^2 = \min_{\widehat{h}} \widehat{h}^{-1} \left(\frac{1}{\log \widehat{h}} \right)^2.$$

To minimise C is now equivalent to maximise the function $\tilde{C}^{-1}$:

$$\tilde{C}^{-1}(\widehat{h}) = (\log \widehat{h})^2 \widehat{h},$$

so, differentiating with respect the step, we get:

$$\frac{d\tilde{C}^{-1}}{d\widehat{h}} = (\log \widehat{h})^2 + 2\widehat{h} \log \widehat{h} \frac{1}{\widehat{h}} = (\log \widehat{h})^2 + 2 \log \widehat{h} = 0 \iff \begin{cases} \log \widehat{h} + 2 = 0 \iff \widehat{h} = e^{-2} \iff h = \frac{\rho}{e^{-2}}, \\ \log \widehat{h} = 0 \iff \widehat{h} = 1 \iff h = \rho. \end{cases}$$

The value $\widehat{h} = 1$ is a minimum of C^{-1} and $\widehat{h} = e^{-2}$ is the maximum that we are looking for. This proves the statement. □

In some cases, and in order not to lose significant digits, it is interesting to use relative errors instead of absolute errors. In these cases the first term is usually the biggest one so the last term should be ε times smaller than the first one in order to be good enough:

$$\left\|x^{(N)}\right\|_\infty h^N \leq \varepsilon \left\|x^{(1)}\right\|_\infty h \implies h \leq \left(\frac{\varepsilon \left\|x^{(1)}\right\|_\infty}{\left\|x^{(N)}\right\|_\infty}\right)^{\frac{1}{N-1}}.$$

If we also want to include the effects of the previous term, we should take into account that the $N-1$ term is larger, therefore, it should be compared with something bigger than the first term. Observe that, if we assume $\|x^{(j)}\| = \frac{\kappa}{\rho^j}$, $c_1 \leq \kappa \leq c_2$, then:

$$\frac{\|x^{(N-1)}\|h^{N-1}}{\|x^{(N)}\|h^N} = \frac{\rho}{h} = e^2,$$

where $\frac{\rho}{h} = e^2$ is the more efficient step size when the error tends to zero, given by Proposition 1. Therefore,

$$\left\|x^{(N-1)}\right\|_\infty h^{N-1} \leq \varepsilon e^2 \left\|x^{(1)}\right\|_\infty h \implies h \leq \left(\frac{\varepsilon e^2 \left\|x^{(1)}\right\|_\infty}{\left\|x^{(N-1)}\right\|_\infty}\right)^{\frac{1}{N-2}}.$$

Then, the optimal step size becomes:

$$h_{\text{opt}} = \min\left\{\left(\frac{\varepsilon e^2 \left\|x^{(1)}\right\|_\infty}{\left\|x^{(N-1)}\right\|_\infty}\right)^{\frac{1}{N-2}}, \left(\frac{\varepsilon \left\|x^{(1)}\right\|_\infty}{\left\|x^{(N)}\right\|_\infty}\right)^{\frac{1}{N-1}}\right\}.$$

2.3.3 Adapting Taylor's method for the propagation of jets

When we implement Taylor's method for the propagation of jets, we substitute the real vectors $\vec{x}$ by polynomials $\vec{P}(\vec{\xi})$ in the state variables. The independent term of $\vec{P}$ coincides with the state variables, $\vec{P}(0) = \vec{x}$, and the flow expansion used for the Jet Transport at the time t_0 and position $\vec{P}(\vec{\xi})$ is

$$\phi(t_0 + h; t_0, \vec{P}(\vec{\xi})) = \sum_i P^{(i)}(\vec{\xi})h^i = \phi(t_0; t_0, \vec{P}(\vec{\xi})) + \frac{\partial\phi(t_0; t_0\vec{P}(\vec{\xi}))}{\partial t}h + \frac{\partial^2\phi(t_0; t_0\vec{P}(\vec{\xi}))}{2\partial t^2}h^2 + \dots$$

The polynomial $\phi(t_0 + h; t_0, \vec{P}(\vec{\xi}))$ can be seen as a polynomial with $l+1$ variables. The first l are the phase space variables, $\vec{\xi}$, and the last one is the temporal variable h. Therefore, if n is the order used in the polynomial algebra, then

$$\phi(t_0 + h; t_0, \vec{x}_0 + \vec{\xi})\big]_m = \sum_{i=0}^{N} \sum_{|k|<n} c_{m,k}^{(i)} \vec{\xi}^k h^i$$

is the expansion of the solution on time and phase space variables provided by Taylor's method.

Hence, the first to be done is to introduce the polynomial algebra arithmetic in the Taylor propagation method. The second is to determine the time step size that we can use at each step. In the Jet Transport, the step size must cover the precision of the higher order terms of the solution, which include the variational terms. Therefore, the idea is to apply the method used in the standard Taylor's method to each of the multi-indices k. Observe that the higher order terms, which are the values of the derivatives up to the desired order, increase their value very fast. Therefore, using the first step size technique, that only uses the lasts terms of the polynomial, will give very small step sizes. However, if at each multi-index we use the relative errors, we are comparing the first term of the jet in time of the corresponding multi-index with the last one in time of the same multi-index k; since both of them are large, the method will give bigger time step sizes.

The implementation of the strategy is as follows.

Let $c_{m,k}^{(N)}$ be the coefficient of exponent of h^N with multi-index k, in the expansion of the m-th component of $\phi(t_0+h;t_0,\vec{x}_0+\vec{\xi})$. Then, for every multi-index k compute:

$$\begin{aligned}\psi_k^N &= \max_{1\le m\le n}|c_{m,k}^{(N)}|,\\ \psi_k^{N-1} &= \max_{1\le m\le n}|c_{m,k}^{(N-1)}|,\\ \psi_k^1 &= \max_{1\le m\le n}|c_{m,k}^{(1)}|.\end{aligned}$$

Analogously to the one dimensional case, compute:

$$\begin{aligned}h_{N,k} &= \left(\frac{\varepsilon\psi_k^1}{\psi_k^N}\right)^{\frac{1}{N-1}},\\ h_{N-1,k} &= \left(\frac{\varepsilon e^2\psi_k^1}{\psi_k^{N-1}}\right)^{\frac{1}{N-2}}.\end{aligned}$$

Finally, the selected step size is:

$$h_{\text{opt}} = \min_k\left\{h_{N-1,k}, h_{N,k}\right\}.$$

Observe that the best step size is still reached by $h=\rho e^{-2}$, since the condition that we used is that the order of the method is N^2. Now, each operation is slower, but the order of the operations is still N^2. Therefore, it still has sense to multiply by the e^2 in the step selection for the orders $N-1$.

Finally, it is interesting to add a security factor r to the time step. Using a security factor for instance $r=0.8$ we reduce the time step doing more integration steps but we increase the precision in which the steps are made.

2.3.4 Numerical results and examples

The first results of the Jet Transport method correspond to the integration of a linear centre. This is an interesting model since we can compute analytically all the variational terms and compare them with the results given by the Jet Transport. The second example is simple pendulum. For this model we can check the precision of the variational terms as well as of the energy, written as a polynomial in the variables of the jet ($\vec{\xi}$). Some results for the perturbed pendulum and the CR3BP will also be shown.

The linear centre

Consider the linear centre defined by:

$$\begin{cases}\dot{x}=y,\\ \dot{y}=-x.\end{cases}$$

The flow associated to the initial condition (x_0,y_0) is $\phi(t;t_0,x_0,y_0)=r_0(\cos(t+\theta_0),\sin(t+\theta_0))$, where $r_0=\sqrt{x_0^2+y_0^2}$, and $\theta_0=\arctan(y_0/x_0)$. Observe that $I=\sqrt{x^2(t)+y^2(t)}=r_0$ is a first integral of the system.

The first variational equations

$$\begin{pmatrix}\dot{x}_{x_0} & \dot{x}_{y_0}\\ \dot{y}_{x_0} & \dot{y}_{y_0}\end{pmatrix}=\begin{pmatrix}y_{x_0} & y_{y_0}\\ -x_{x_0} & -x_{y_0}\end{pmatrix},\begin{pmatrix}x_{x_0}(0) & x_{y_0}(0)\\ y_{x_0}(0) & y_{y_0}(0)\end{pmatrix}=\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix},$$

have as solution

$$\begin{cases}x_{x_0}(t)=y_{y_0}(t)=\cos(t),\\ x_{y_0}(t)=y_{x_0}(t)=\sin(t).\end{cases}$$

Figure 2.3 shows the variations of the value of the first integral I during 100 time units. As it is seen, the variations are of the order of the machine precision.

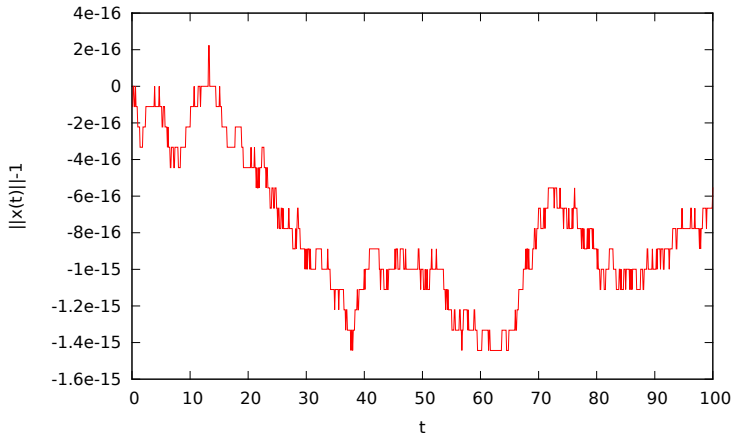


Figure 2.3: For an initial condition in the unit circle $I^2 = x_0^2 + y_0^2 = 1$ of the linear centre, the figure shows the variations of the value of the first integral I, computed using the Jet Transport, during 100 time units.

Figure 2.4 shows the error of the term x_{x_0} of the first variational equations, this is, the difference between the values of $x_{x_0}(t)$, computed using the Jet Transport, and $\cos(t)$

Again the error is of the order of the machine precision. However, when we increase the time interval, the errors increase, while in the propagation of the solution (the zero order terms) the errors was increasing much slower. This fact is usual in the Jet Transport procedures. The higher order terms increase their errors much faster than the lower order ones.

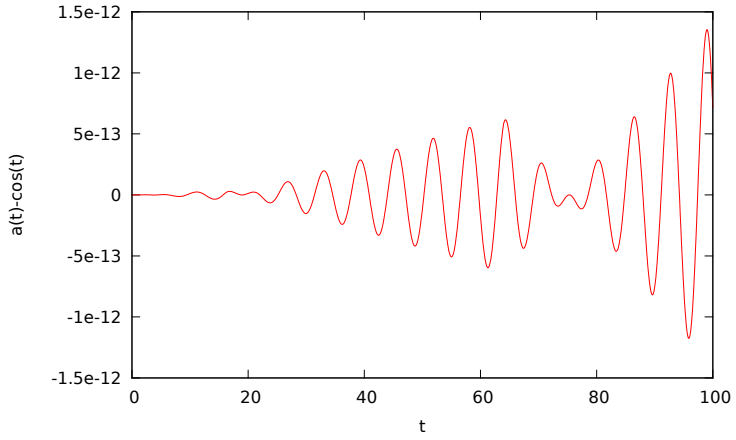


Figure 2.4: For the linear centre, with an initial condition on the unit circle, error of first component of the state transition matrix during 100 time units.

The simple pendulum

Recall that the equations of motion of the simple pendulum can be written as:

$$\begin{cases} \dot{x} = y, \\ \dot{y} = -\sin(x), \end{cases}$$

for which the energy $E(x, y) = \frac{y^2}{2} - \cos(x)$ is a first integral of the above system.

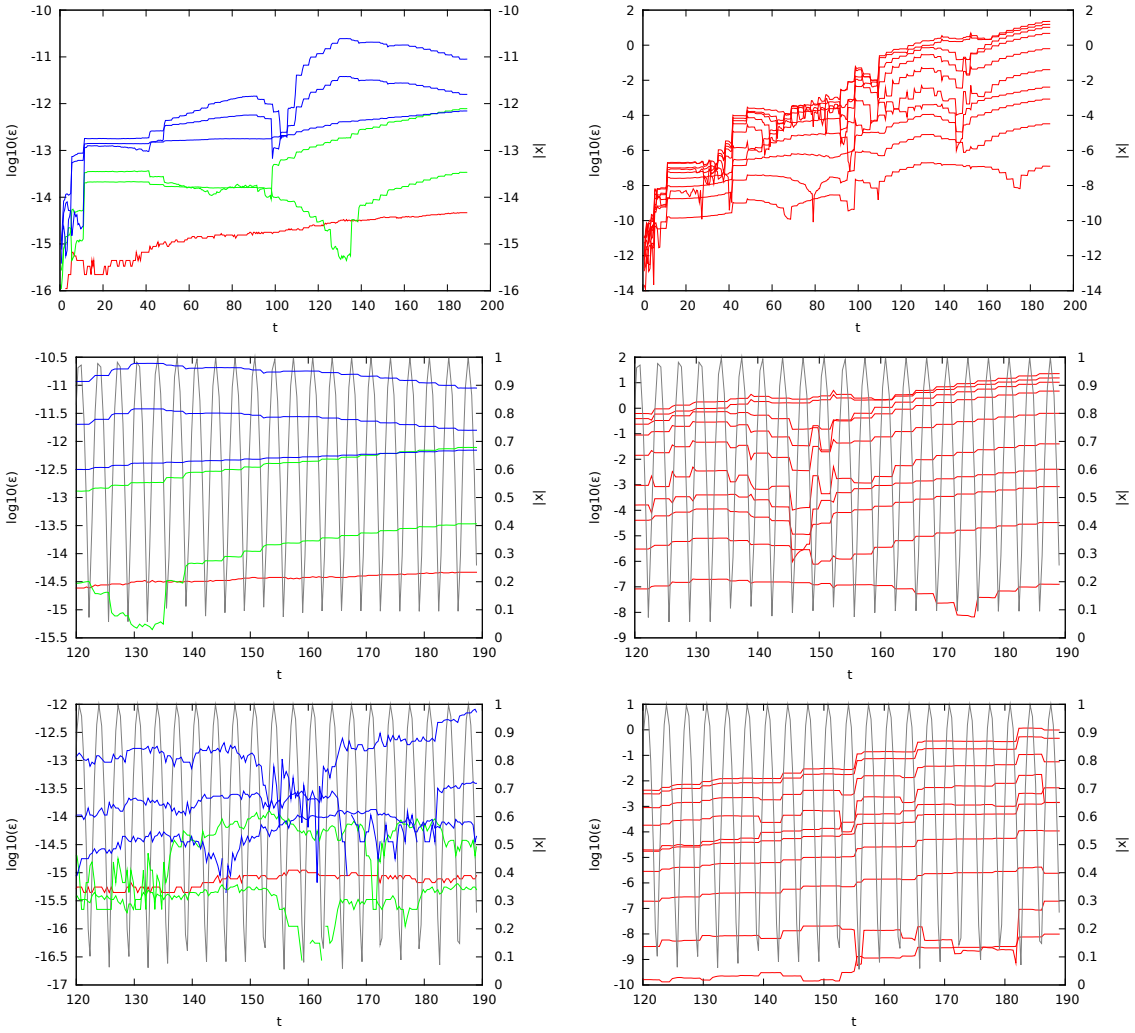


Figure 2.5: This figure shows the behaviour of the error of the Jet Transport method applied to the simple pendulum. The second row is a magnification of the plots of the first row for $t \in [120, 190]$, and the third one shows the results, in this same interval, when a security factor $r = 0.8$ is used for the time step. The grey oscillating curves that appear in the four bottom plots display the absolute value of the angle x along the orbit. The left hand side plots display the error of the independent term (red curves), the two first order terms (green) and the three second order terms (blue) of the energy polynomial evaluated along the trajectory of the pendulum. The right hand side plots display the behaviour of the error of the ten 9-th order terms of the energy polynomial.

Using the Jet Transport for the numerical integration, the top-left plot of Figure 2.5 shows the variations of the coefficients of the expansion of E. The red curve corresponds to the independent term of the. As in the case of the linear center, the error is of the order of the machine precision. We remark that this red curve could also be obtained using just Taylor's method. If we want to see the Jet Transport playing a role we must go to higher order terms.

To test these terms we have used the fact that, since the energy of the system is conserved, all the terms of its expansion must also be conserved. This means the following: let

$$E(x_0+\xi_1, y_0+\xi_2) = \sum_{k\in\mathbb{N}^2} a_k \xi_1^{k_1}\xi_2^{k_2},$$

be the Taylor expansion of E around a certain initial condition (x_0, y_0). $E(x_0+\xi_1, y_0+\xi_2)$ can be obtained, by means of the polynomial algebra for all the operations involved, computing $E(x, y)$ at the initial conditions of the (polynomial) form: $(x, y) = (x_0 + \xi_1 + 0\cdot\xi_2, y_0 + 0\cdot\xi_1 + \xi_2)$. On the other hand, the energy can also be computed using the polynomial $P_{\vec{x}_0}(\vec{\xi}) = (P_{\vec{x}_0}(\vec{\xi})]_1, P_{\vec{x}_0}(\vec{\xi})]_2)$ obtained with the Jet Transport for the propagation of $\vec{x}_0$ up to a certain epoch t, and evaluating the energy E at these points: $E(P_{\vec{x}_0}(\vec{\xi})]_1, P_{\vec{x}_0}(\vec{\xi})]_2)$. Since the energy must be preserved, the coefficients of this second polynomial must be time-independent and agree with those obtained for $E(x_0+\xi_1, y_0+\xi_2)$.

Figure 2.5 shows the differences between the values obtained for the coefficients a_k of $E(x_0+\xi_1, y_0+\xi_2)$ computed using the two strategies just mentioned. The last row of this figure shows the results when the Jet Transport propagation is done using a security factor $r = 0.8$ in the in the step size control, which means that one the step size has been computed according to what has been explained in the preceding section, it is multiplied by 0.8 to get the final step size. Of course, since the step sizes are smaller the integration results are better.

Direct test of the variational equations

To test the results obtained for the variational equations one can use numerical differentiation for the coefficients of the polynomial expansion of the flow provided by the Jet Transport. Nevertheless, it must be taken into account that the coefficients of order k do not give the k-th variational terms. With the Jet Transport, the polynomial expansion that is obtained is:

$$\phi(t; t_0, x_0+\delta x) = \sum_{k\in\mathbb{N}^l} \overline{a}_k \delta x^k,$$

while, if the variational equations are used we obtain the differentials of the solution with respect to the initial conditions $\mathrm{D}^i\phi(t; t_0, x_0), \quad \forall i \in \mathbb{N}$, and the polynomial expansion as function of δx is then

$$\phi(t; t_0, x_0+\delta x) = \sum_{i=0}^{\infty} \frac{1}{i!}\mathrm{D}^i\phi(t; t_0, x_0)\delta x.$$

Due to Schwarz' theorem, in the variational equations there are some coefficients that are equal, therefore:

$$\mathrm{D}^i\phi(t; t_0, x_0)\delta x = \sum_{\substack{k\in\mathbb{N}\\|k|=i}} \frac{i!}{k_1!\ldots k_l!} a_k \delta x^k.$$

Hence, the polynomial approximation of the solution using the variational equations is:

$$\phi(t; t_0, x_0+\delta x) = \sum_{i=0}^{\infty}\frac{1}{i!}\sum_{\substack{k\in\mathbb{N}^l\\|k|=i}} \frac{i!}{k_1!\ldots k_l!} a_k\delta x^k = \sum_{k\in\mathbb{N}^l}\frac{1}{k_1!\ldots k_l!}a_k\delta x^k.$$

So, for the solutions of the variational equations the following relation holds:

$$\overline{a}_k = \frac{1}{k_1!\ldots k_l!}a_k \tag{2.9}$$

With this in mind, to test the computation of the variational terms the following numerical derivation formula can be used:

$$a_k \approx \frac{a_{k-e_i}(t; t_0, x_0 + \delta e_i) - a_{k-e_i}(t; t_0, x_0 - \delta e_i)}{2\delta}, \tag{2.10}$$

where $e_i = (0, ..., 0, i, 0, ...0)$.

Figure 2.6 shows the differences between the computations of the variational terms using the approximation given by (2.10), for several values of δ, and the values obtained using the Jet Transport for the first and second variational terms. One can see that for all the cases the behaviour of the error is the expected: it behaves like δ^2 until a certain value of δ is reached, from that value on the round off error becomes the dominant contribution to the error. Using Richardson's extrapolation the results can be improved.

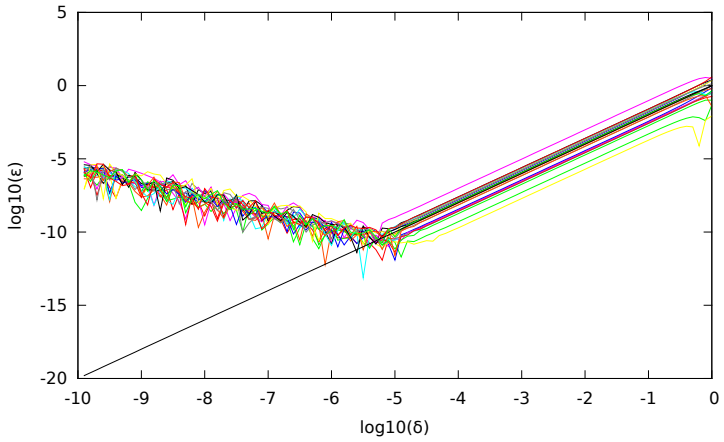


Figure 2.6: For the pendulum, value of the difference between the first variational terms using the Jet Transport and using numerical derivatives in function of the derivation step size δ.

The size of the box around the initial condition

Let us take a look to the size of the maximum box that we can take around an initial condition and what happens for larger boxes. The computations have been done for the pendulum using all the variational terms up to order 9.

Figure 2.7 shows the propagation, after 189 time units, of square boxes around the initial condition $(x_0, y_0) = (1, 0)$ of different diameters $\delta = 10^{-3}$, 10^{-2} and 10^{-1}. In all the plots, the initial box is in blue and its image after 189 time units in green; the two red crosses at the center of the initial box and its image are the points (x_0, y_0) and $\phi(189; 0, (1, 0))$, respectively; the orbit of (x_0, y_0) , $\phi(t; 0, (1, 0))$, is the black curve; the orbits of the innermost $(x_0, y_0) = (1 - \delta, 0)$ and outermost $(x_0, y_0) = (1 + \delta, 0)$ points of the box a shown in grey.

For small size boxes the propagation is good; however, for the medium size box there are some points in the boundary that start to be outside the region delimited by the grey band inside which the propagated box should be. Finally, if $\delta = 10^{-3}$ we can see that there are some points of the image of the initial box outside the range. This happens because the order 9 polynomial $P(\vec{\xi})$ used to compute $\phi(t; t_0, \vec{x}_0)$ is not able to approximate properly the solution. To get better approximations we should take higher orders of the polynomial.

To get good final regions of propagation, it is also important where the initial condition is located. Initial conditions close to an hyperbolic structure, like an hyperbolic invariant manifold, can be propagated only using smaller boxes while those ones in regular regions can be propagated using bigger boxes. This is shown clearly in Figure 2.8, where the size of the box is kept fixed but the initial point $\vec{x}_0$ approaches the invariant manifolds.

The same kind of effects happen in the periodically perturbed pendulum defined by the equations:

$$\ddot{x} = \left(\frac{5}{2}\cos(5t) - 1\right)\sin(x).$$

Figure 2.9 shows the image of an initial box of size 0.001×0.001 for different initial conditions. We can see that the larger ones are close to the invariant manifolds (shown in grey) while those ones that are far away from the manifolds are small. This property will be used in Chapter 3 to give dynamical indicators of regularity of the system using the Jet Transport.

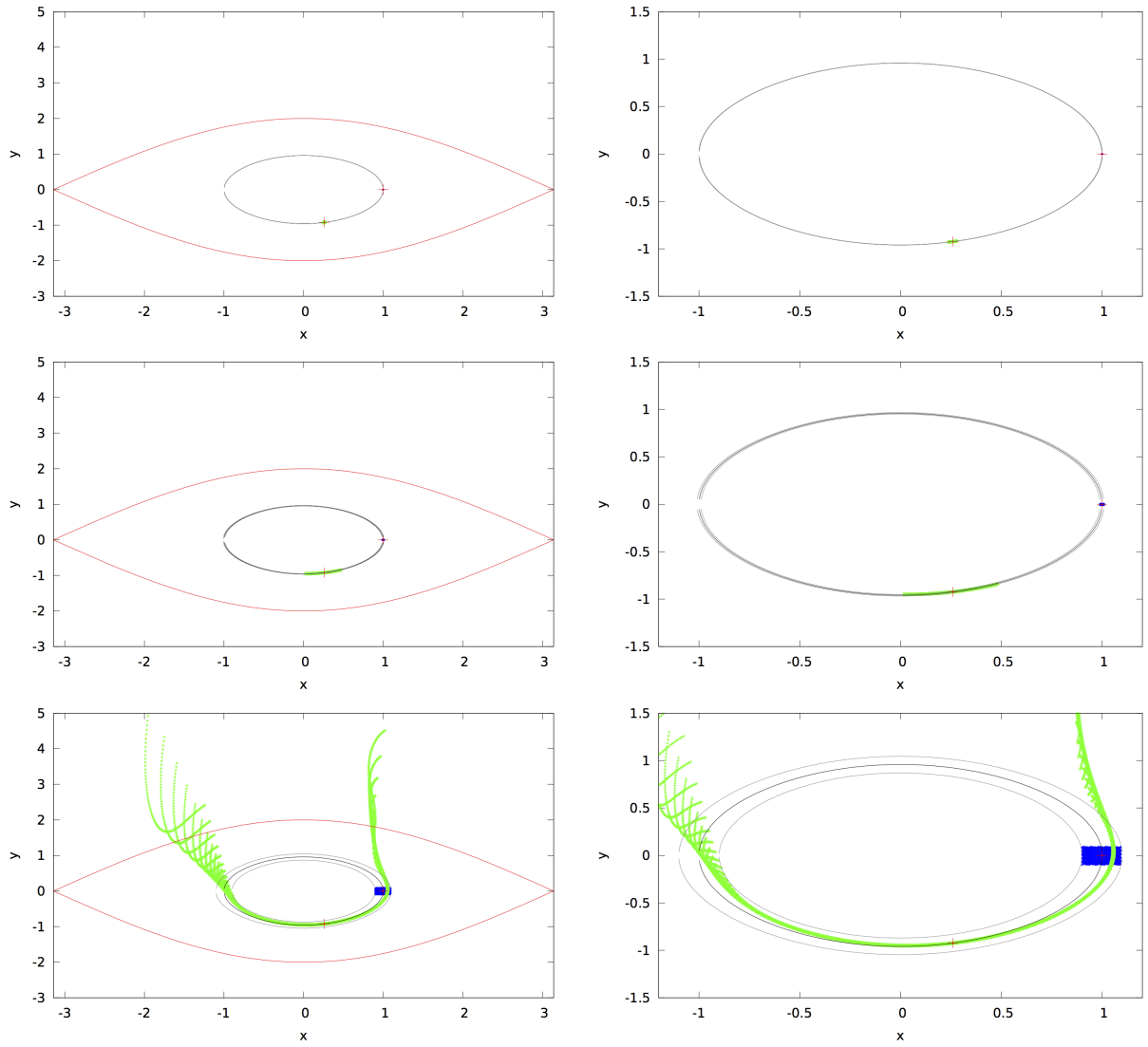


Figure 2.7: Propagation up to time 189 of some boxes with different initial sizes (0.001×0.001, 0.01×0.01 and 0.1×0.1 for the first, second and third rows, respectively). The propagation is done using the variational terms up to order 9. Both, the initial (blue) and the final (green) box are plotted as well as the invariant manifolds of the hyperbolic points (red), the orbit of the centre of the box (black) and the inner and outer orbit of the points inside the box(grey). The outer red curves are the invariant manifolds of the unstable equilibrium point. The right hand side plots are enlargements of the left hand side ones.

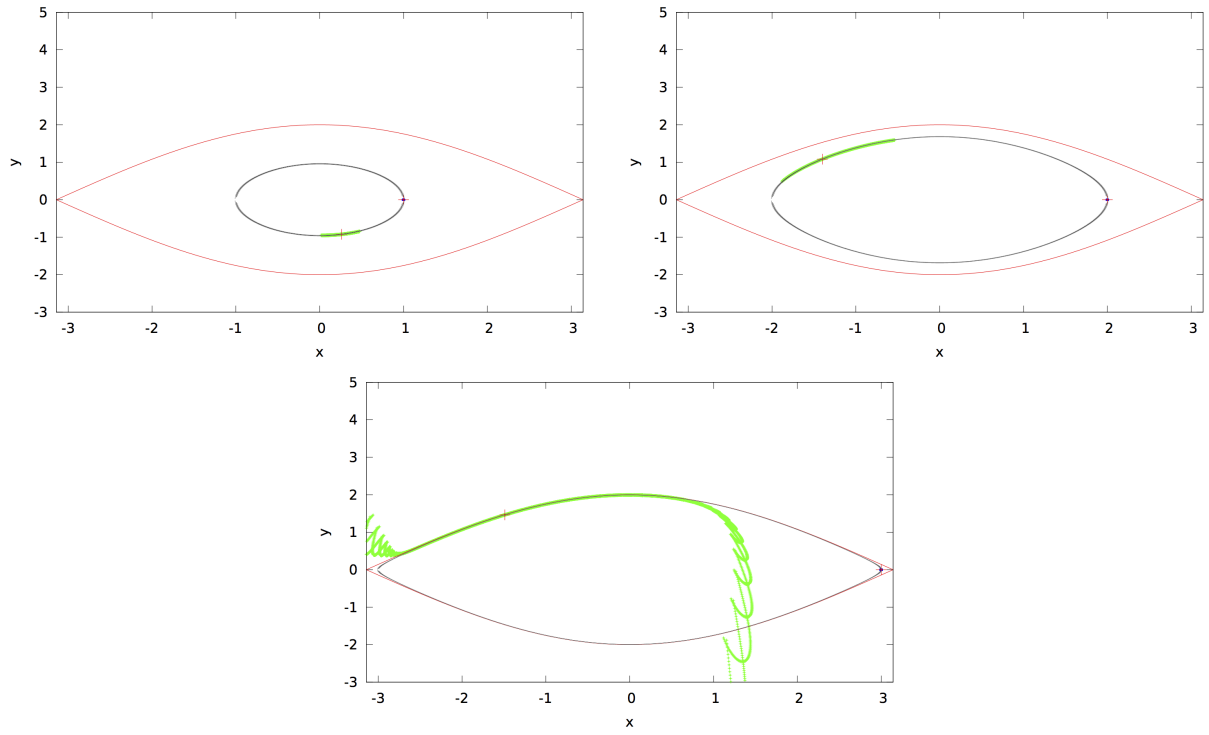


Figure 2.8: For the simple pendulum, propagation up to time 189 of 0.01×0.01 box centred (from top to bottom, left to right) at (1,0), (2,0), (3,0) using the variational terms up to order 9. Both, the initial (blue) and the final (green) box are plotted as well as the invariant manifolds of the hyperbolic points (red), the orbit of the centre of the box (black) and the inner and outer orbit of the points inside the box(grey).

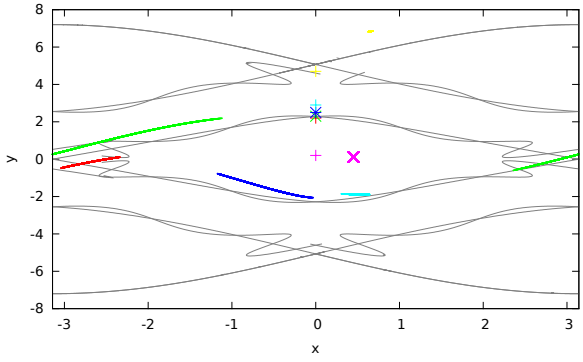


Figure 2.9: For the perturbed pendulum, propagation up to $t = 20$ of a 0.001×0.001 box centred at different initial conditions along the $x = 0$ axis, and using the variational terms up to order 9. The colour crosses are the initial conditions corresponding to the final box of the same colour.

The CR3BP

The last model that has been used to test the flow propagation with Jet Transport is in the CR3BP, that has already been described.

The first test done to check the precision of the Jet Transport computations is related to the accuracy of the derivatives of the flow map obtained by the Jet Transport compared with the values obtained by numerical differentiation using:

$$f'(x) \approx \frac{f(x+\delta) - f(x-\delta)}{2\delta}.$$

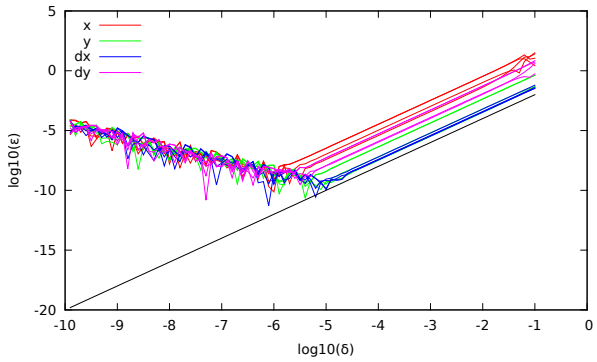


Figure 2.10: For the CR3BP, accuracy tests for the computation of the variational equations using the Jet Transport. The figure shows, after 100 time units and initial conditions $(x, y, \dot{x}, \dot{y}) = (0.9, 0.3, 0.1, 0.4)$), the differences between the first order variational equations computed with the Jet Transport and the values obtained using numerical derivatives with step size δ, for each one of the four variables of the system.

Figure 2.10 shows the differences between the first order derivatives, which are related to the first order variational equations, obtained from the Jet Transport and the numerical values of the variational equations computed using the previous differentiation formula. As usual for this kind of numerical computations, when we decrease the value of δ the errors become smaller until a certain value of δ from which the round-off error becomes larger than the truncation error. To improve this results a Richardson's extrapolation method has been used, according to the following recursive formulae:

$$\begin{aligned} F_1(h) &= f'(h), \\ F_{i+1}(h) &= F_i(h) \frac{F_i(h) - F_i(2h)}{2^{i+1} - 1}. \end{aligned}$$

Figure 2.11 shows the results of five steps of the extrapolation method for the first variable x of the CR3BP. From this figure, it is clear that at each extrapolation step the errors decrease and that the approximation given by the Jet Transport is very good (the error is of the order of 10^{-13} using double precision arithmetic).

The last test that has been done is related to the conservation of a first integral. As before, instead of checking its conservation as a scalar value, we check it as a polynomial. Analogously to the case of the pendulum, we compute a polynomial giving the variations of the values of the first integral associated to small deviations of the variables. Then, at some point during an integration, we compute again the polynomial first integral and check its components against the initial values. Figure 2.12 shows the difference between the first, second and third order terms of the polynomial first integral along the integrations of the CR3BP differential equations. As expected, the longer the time the bigger the error, but always within a good accuracy.

The above results ensure that it is possible to extract information about the high order variational terms from the output of the Jet Transport computations. The main drawback of the Jet Transport is that, in case

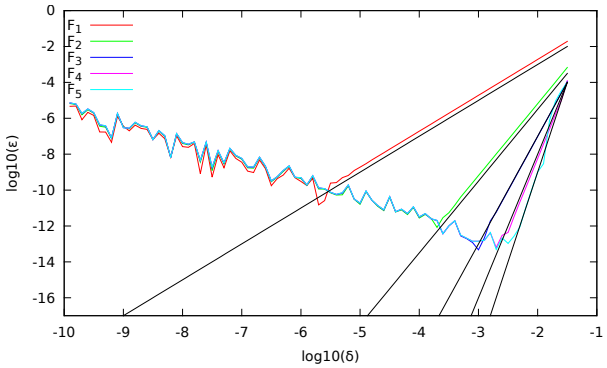


Figure 2.11: Accuracy test for the first order variational terms in the CR3BP after 2 time units. The colour curves display the difference between the first order variational equations of the first coordinate ($\frac{\partial \phi^T_{t_0}(x_0)]_1}{\partial x}$) computed with the Jet Transport, and the values obtained using Richardson's extrapolation step F_i as a function of δ in $\log_{10}$ scale. The black lines show the trends expected according to the order of the formula.

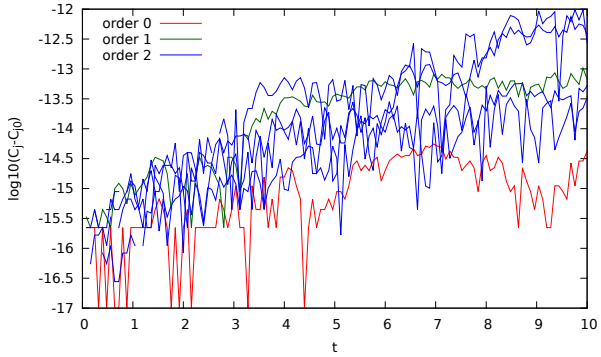


Figure 2.12: Accuracy tests for the conservation of the first integral of the CR3BP. As a function of time, the figure displays the differences between the initial energy polynomial, with initial conditions $(x, y, \dot{x}, \dot{y}) = (0.9, 0.3, 0.1, 0.4)$, and the one computed using the jet for the independent term (in red), the first order terms (in green) and second order terms (in blue).

that some information is needed for a single point, computations will be slow compared to a single standard integration. However, if we are looking for information of neighbouring points, as is the case of many Monte Carlo simulations, or we need high order variational terms, then the Jet Transport becomes useful. Figure 2.13 shows the amount of CPU time needed to integrate different sets of initial conditions of the CR3BP, during 20 time units, in front of the CPU time required by the Jet Transport to get the images under the flow of the same sets of initial conditions and for the same time interval. We can appreciate the almost constant behaviour of the Jet in front of the linear growth using a RK78 classical procedure.

In the CR3BP it is usual to use the first variational equations to determine stability of objects, for instance the stability of periodic orbits. To compute them we only need the jet to order one. As we stated previously, one can develop specific functions of the polynomial algebra such that the jet of order one can be computed faster than using the general developed routines.

A last velocity test, that compares the three methods that we have available (the variational equations, using a general implementation and using a specific order one implementation), is shown in table 2.4. The table shows the time required to integrate 1000 times 1 unit of time for two different initial orbits. The first one is a planar Lyapunov orbit around L_1, the second is an orbit between the two primaries that passes close to a singularity, this entails that much more CPU time will be required to compute the final result. The

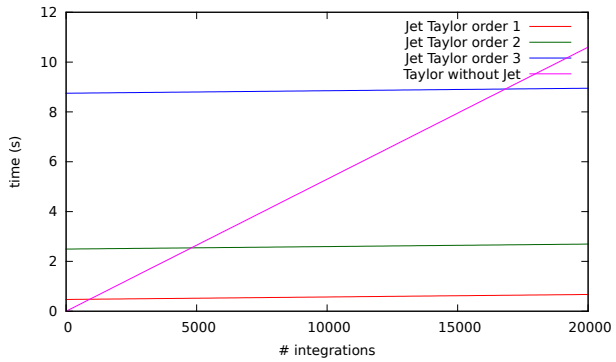


Figure 2.13: In the CR3BP, CPU Time in seconds spent as a function of the number of initial conditions propagated using the Jet-Taylor method with orders 1,2,3 (respectively red, green and blue) and a usual Taylor Method without jet propagation (magenta). Each integration is done in an orbit randomly chosen close to the initial condition at $(1, 0, 0)$ with initial velocity $v_0 = (1, 1, 1)$ during 20 adimensional time units. The mass parameter is $\mu = 0.012$. The computations have been carried out with an Intel Xeon(R), CPU E5645, 2.40GHz.

	Jet (specific)	Jet (generic)	Variational equations
time 1st orbit	0m2.580s	0m31.486s	0m0.864s
integration steps 1st orbit	15	15	14
time 2nd orbit	0m29.910s	6m23.060s	0m9.513s
integration steps 2nd orbit	174	174	145

Table 2.4: CPU time spent by three methods of computing the first variational equations. The first column shows the time spent using a specific implementation of the polynomial algebra functions at order one, the second using a generic implementation and the last one using variational equations. The first two orbits refers to an initial condition in a planar Lyapunov orbit around L_1 while the last two refers to an orbit between the two primaries with a close approach to one of them.

table also shows the number of integration steps needed. We can see that using the variational equations give the fastest results, then comes the Jet Transport using specific implemented routines to deal with an specific order of the jet (order 1), and finally the general routines implemented in the polynomial algebra.

2.4 Poincaré maps

In this section we will give some procedures to compute Poincaré maps using Jet Transport.

Consider a manifold $\mathcal{M}$ defined in the phase space of a certain dynamical system defined by $\dot{\vec{x}} = f(t, \vec{x})$. A Poincaré map on $\mathcal{M}$ is a map defined from the manifold to itself such that gives the position of a point on the manifold when, under the flow ϕ of the system, the orbit of this point crosses again the manifold. A necessary condition to properly define a Poincaré map is the *transversal condition*, this is: that the manifold cannot be tangent to the flow.

The Poincaré map $P_{\mathcal{M}}$ is defined as:

$$P_{\mathcal{M}}(\vec{x}_0) = \phi(T; t_0, \vec{x}_0) \quad \text{with } T \text{ s.t.} \quad \phi(T; t_0, \vec{x}_0) \in \mathcal{M} \text{ and } T > 0 \text{ is minimum.}$$

In some cases it is convenient to compute not the first crossing of the flow with $\mathcal{M}$ but the first crossing in a certain direction. A typical condition of this kind is to require to angle between the direction at the crossing and the starting direction to be smaller than $\pi/2$.

In the description that follows, we will consider that $\mathcal{M}$ is the hyper-plane defined by $\mathcal{M} = \{\vec{x} \in \mathbb{R}^n \mid \vec{x}]_n = C\}$, i.e. the last coordinate of the points in $\mathcal{M}$ is constant; of course, it is possible to take more generally defined hyper-planes as will be seen later.

Using standard floating point arithmetic, the procedure for the computation of the Poincaré map iterates of points in $\mathcal{M}$ is as follows:

- Starting at $\vec{x}_0 \in \mathcal{M}$, compute the orbit of this point under the flow, using any numerical integration method, until it crosses again $\mathcal{M}$ in the same direction as $\vec{x}_0$, i.e. if $\vec{x}_m$ and $\vec{x}_{m-1}$ are the last two points along the orbit of $\vec{x}_0$ provided by the integration method, they must fulfil:

$$\begin{aligned}(\vec{x}_m]_n - C)(\vec{x}_{m-1}]_n - C) < 0,\\ (\vec{x}_m]_n - \vec{x}_{m-1}]_n)\dot{\vec{x}}_0]_n > 0.\end{aligned}$$

- Refine the value of the time required to reach the section. To do so, we can use Newton's method for the last component of the vector-field. If t_m is such that $\phi(t_m; t_0, \vec{x}_0)] = \vec{x}_m$, the condition to be in the Poincaré section is that $\phi(t_m + \sigma; t_0, \vec{x}_0)]_n - C = 0$. Taking $\sigma_0 = 0$ and applying Newton's method we get:

$$\sigma_1 = \sigma_0 - \frac{\phi(t_m + \sigma_0; t_0, \vec{x}_0)]_n - C}{\dfrac{d\phi(t_m + \sigma_0; t_0, \vec{x}_0)]_n}{d\sigma}} = \sigma_0 - \frac{\phi(t_m + \sigma_0; t_0, \vec{x}_0)]_n - C}{f(t_m + \sigma_0, \vec{x}_m)]_n}.$$

  The procedure is iterated until the last component of $\phi(t_m+\sigma_n; t_0, \vec{x}_0)$ equals C within a given tolerance, i.e. $|\phi(t_m + \sigma_n; t_0, \vec{x}_0)]_n - C| < \varepsilon$.

- If we are interested in the computation of the differential of the Poincaré map, $DP_{\mathcal{M}}$ we can use the variational equations to integrate the extended system, in such way that we obtain: $\phi(T; t_0, \vec{x}_0) = \vec{x}_{\mathcal{M}} \in \mathcal{M}$ and $A = D\phi(T; t_0, \vec{x}_0)$. Observe that A is not the differential of the Poincaré map, it is the differential of stroboscopic map –the map at time T– and, in general, it is not true that the neighbouring points of $\vec{x}_0$ will need the same amount of time to reach $\mathcal{M}$. Therefore, some small corrections are needed to reach the final position in $\mathcal{M}$.

  Assuming that all the values are close enough, we can use a first order correction. This is, we will look for a Δt such that the last component of

$$\vec{x}_{\mathcal{M}} + A\vec{\xi} + \mathcal{O}(|\vec{\xi}|^2) + \Delta t f(T, \vec{x}_{\mathcal{M}}),$$

  is equal to C. Or equivalently:

$$(A\vec{\xi})]_n + \Delta t f(T, \vec{x}_{\mathcal{M}})]_n = 0.$$

  Isolating Δt, and using the transversal condition, we get:

$$\Delta t = -\frac{(A\vec{\xi})]_n}{f(T, \vec{x}_{\mathcal{M}})]_n}.$$

  Therefore, the final differential correction will be:

$$A\vec{\xi} - \frac{(A\vec{\xi})]_n}{f(T, \vec{x}_{\mathcal{M}})]_n} f(T, \vec{x}_{\mathcal{M}}),$$

  which is equivalent to a differential matrix $\tilde{A}$ with components equal to

$$\tilde{A}_{ij} = A_{i,j} - A_{n,j}\frac{f^{(i)}(T, \vec{x}_{\mathcal{M}}))}{f^{(n)}(T, \vec{x}_{\mathcal{M}}))}.$$

In what follows we will discuss how to adapt the Jet Transport to the Poincaré map computations. Additional information can be found in [GMB05, GBM06].

2.4.1 Computation of Poincaré maps using the Jet Transport

As it is usual in the Jet Transport procedures, the first thing that must be done is to get the zero order solution accurately. To that end, we can apply the previous explained method for floating point arithmetics. In this way, we find $\vec{x}_{\mathcal{M}}$ and T such that $\phi(T; t_0, \vec{x}_0) = \vec{x}_{\mathcal{M}} \in \mathcal{M}$, therefore, the propagation of $\vec{x}_0$ using the Jet Transport will determine a polynomial $P(\vec{\xi}) = \vec{x}_{\mathcal{M}} + \sum_{1 \leq |k| \leq n} a_k \vec{\xi}^k$.

Even if the values of $\vec{\xi}$ are such that $\vec{x}_0 + \vec{\xi} \in \mathcal{M}$, it is not true that the values of $P(\vec{\xi})$ will be in $\mathcal{M}$. For this reason we have to modify the values of a_k in P in order to get the Poincaré mapping $P_{\mathcal{M}}$. This is done computing a modified final time $\tau(\vec{\xi}) = T + \sum_{1 \leq |k| \leq n} b_k \vec{\xi}^k$ such that $\phi(\tau(\vec{\xi}); t_0, \vec{x}_0 + \vec{\xi}) \in \mathcal{M}$ for all $\vec{\xi}$.

Consider the polynomial $P(\vec{\xi}, \tau)$ as the Taylor expansion in both the time variable t and the space variable $\vec{x}$ of the flow map $\phi(T + \tau; t_0, \vec{x}_0 + \vec{\xi})$. It is obtained using the propagation in time of the flow presented in Section 2.3. The polynomial $P(\vec{\xi}, \tau)$ is a polynomial in one variable (time) which coefficients are polynomials in l variables, i.e. the coefficient of τ^i is given by the polynomial $P_i(\vec{\xi}) = \sum_k a_{i,k} \vec{\xi}^k$. We want to find $\tau(\vec{\xi})$ such that the following section condition is fulfilled:

$$\left(\sum_{i=0}^{l} \left(\sum_{0 \leq |k| \leq n} a_{i,k} \vec{\xi}^k \right) \tau^i(\vec{\xi}) \right) \Bigg]_n - C = 0. \tag{2.11}$$

The coefficients $a_{i,k}$ of this equation are known, and we want to determine the coefficients b_k of $\tau(\vec{\xi})$ such that:

$$\left(\sum_{i=0}^{l} \left(\sum_{0 \leq |k| \leq n} a_{i,k} \vec{\xi}^k \right) \left(\sum_{1 \leq |k| \leq n} b_k \vec{\xi}^k \right)^i \right) \Bigg]_n - C = 0. \tag{2.12}$$

Observe that $\tau(\vec{\xi}) = \sum_{|k| < n} b_k \vec{\xi}^k$ gives the additional (positive or negative) time that the initial condition $\vec{x}_0 + \vec{\xi}$ needs to reach again the section, therefore $h(0) = 0$, since if we depart from $\vec{x}_0$ we do not need any additional time.

Now, we have different options to determine the coefficients b_k:

- Use Newton's method for jets to solve and find the value of τ.
- Use iterative formulae to compute directly the values of b_k.

An easy example

Assume that $\mathcal{M}$ is defined by $G(\vec{x}) = \vec{x}]_i = C$. Recall that Euler's method applied to polynomials can be written as

$$\vec{x}_{n+1} = P_{t_n, x_n}(\vec{\xi}) + h f(P_{t_n, x_n}(\vec{\xi})).$$

In our case, this gives the following condition:

$$\Big(P(\vec{\xi}) + \Delta t(\vec{\xi})\, f(P(\vec{\xi})) \Big) \Big]_i = C.$$

Therefore, a first approximation of the polynomial $h(\vec{\xi})$ is:

$$h(\vec{\xi}) = \Delta t(\vec{\xi}) = \frac{C - P(\vec{\xi}) \Big]_i}{f(P(\vec{\xi})) \Big]_i}.$$

Observe that this method does not require the computation of the differential of the Poincaré map. Indeed, the values of $\tau(\vec{\xi})$ obtained in that way will be the first order coefficients of the next procedure.

Using Newton's method

We can apply Newton's method to obtain the solution of the equation (2.11) using as initial seed $t_0 = 0$ and the recursion

$$t_{j+1} = t_j - \frac{\sum_{i=0}^{l} \left(\sum_{0 \leq |k| \leq n} a_{i,k} \vec{\xi}^k \right) t_j^i}{\sum_{i=0}^{l-1} (i+1) \left(\sum_{0 \leq |k| \leq n} a_{i+1,k} \vec{\xi}^k \right) t_j^{i-1}}.$$

Observe that the numerator is exactly the function that we want to solve and the denominator is the derivative the function, that only depends on one variable, and since it is a polynomial is easy to compute.

The values t_j depend on $\vec{\xi}$, in fact, they are polynomials in $\vec{\xi}$, since at each step we are computing the i-th power of the polynomial t_j multiplied by the P polynomial and divided by another polynomial in $\vec{\xi}$.

After several iterations t_j will be the polynomial that we are looking for. In order to find the polynomial $P_{\mathcal{M}}$ we need to do the composition between the last polynomial, $t_j \tau(\vec{\xi})$, and the polynomial providing the Jet Propagation, like in equation 2.11. In section 2.6 we will see that this kind of Newton's methods double the precision of the algorithm, at each iteration the order of the coefficients correctly computed is multiplied by two.

A general iterative procedure

The second option that we have for the computation of the function $\tau(\vec{\xi})$, is to implement an iterative procedure similar to the one of Section 2.2.

In order to simplify the notation, let $P_{i,j}$ be

$$P_{i,j} = \sum_{|k|=j} a_{i,k} \vec{\xi}^k,$$

this is, $P_{i,j}$ contains the coefficients of degree i in time and j in $\vec{\xi}$. In a similar way, let H_n^i be

$$H_n^i = \left(\sum b_j \vec{\xi}^j \right)^i \Bigg]^{(n)},$$

the coefficient of order n in $\vec{\xi}$ of $\tau(\vec{\xi})^i$. For notation convenience the order in $\vec{\xi}$ can be a natural number n or a multi-index k. In the first case it means all the elements whose order is n, while in the second refers to the element with multi-index exactly k.

Using this notation, equation (2.12) becomes:

$$\sum_{i=0}^{l} \left(\sum_{j=0}^{n} P_{i,j} \right) \left(\sum_{j=0}^{n} H_j^i \right) - C = 0. \tag{2.13}$$

Observe that the elements of the form $P_{i,j}$ are already known, while all the unknowns are in H_j^i. However, we have some knowledge about some terms of H_j^i:

- $H_0^0 = 1$ and $H_j^0 = 0$ for $j \neq 0$. Indeed, $\tau(\vec{\xi})^0 = 1$, therefore, all the terms of degree greater than zero are zero, and the term of order zero is 1.

- $H_j^i = 0$ if $j < i$. Indeed, since $\tau(0) = 0$, we have that the i-th power of h will not contain elements of order smaller than i. Observe that this is the key point of all the procedure that follows. Without this fact it will not be possible to compute recursively all the elements b_j.

- If we know $\tau(\vec{\xi})$ up to order n, then the terms H_j^i with $i < j < n$ are already known. Indeed:

$$\begin{aligned} H_j^i &= \left(\sum_k b_k \vec{\xi}^k\right)^i \Bigg]^{(j)} = \left(\sum_{|k|\le n} \underbrace{b_k}_{\text{known}} \vec{\xi}^k + \sum_{|k|>n} \underbrace{b_k}_{\text{unknown}} \vec{\xi}^k\right)^i \Bigg]^{(j)} \\ &= \sum_{l=0}^{i} \binom{i}{l} \left(\sum_{|k|\le n} b_k \vec{\xi}^k\right)^{i-l} \left(\sum_{|k|>n} b_k \vec{\xi}^k\right)^l \Bigg]^{(j)} \\ &= \left(\sum_{|k|\le n} b_k \vec{\xi}^k\right)^i + \sum_{l=1}^{i} \binom{i}{l} \left(\sum_{|k|\le n} b_k \vec{\xi}^k\right)^{i-l} \left(\sum_{|k|>n} b_k \vec{\xi}^k\right)^l \Bigg]^{(j)}. \end{aligned}$$

  Observe that in the last expression, the term of smaller order is at least of order $n+1$, therefore, if we are looking for terms of order j smaller than n all of them will be in the first term, that only contains known terms.

  Equation (2.13) can be rewritten as:

$$\sum_{l=0}^{n} \sum_{i=0}^{l} \sum_{|j|+|k|=l} P_{i,j} H_k^i = 0.$$

With this notation, all the terms of a given order are selected by the index l of the first sum.

Looking to the terms b_j order by order, and using the known information described above, we have that:

- The independent term is $b_0 = 0$, by hypothesis.
- Collecting terms of order one in $\vec{\xi}$, we get:

$$\begin{array}{ll} \text{for } i = 0: & P_{0,1}\, 1 \\ \text{for } i = 1: & P_{1,0}\, \underline{H_1^1} \end{array}$$

  The only terms that remain unknown are the ones underlined. If we expand, we obtain:

$$\sum_{|k|=1} a_{0,k} \vec{\xi}^k \cdot 1 + a_{1,0} \cdot \sum_{|k|=1} \underline{b_k} \vec{\xi}^k = 0,$$

  and using the terms with the same multi-index we get:

$$b_k = \frac{-a_{0,k}}{a_{1,0}} \quad \text{for } |k| = 1.$$

- Collecting terms of order two in $\vec{\xi}$, we get:

$$\begin{array}{ll} \text{for } i = 0 & P_{0,2}\, 1 \\ \text{for } i = 1 & P_{1,0}\, \underline{H_2^1} + P_{1,1}\, H_1^1 \\ \text{for } i = 2 & P_{2,0}\, \underline{H_2^2} \end{array}$$

  Operating as before, we get:

$$P_{0,2} \cdot 1 + \underbrace{P_{1,0}}_{a_{1,0}} \underbrace{H_2^1}_{b_j\text{'s}} + \sum_{|j|+|k|=2} P_{1,k} H_j^1 + \sum_{|j|+|k|=2} P_{2,j} H_k^2 = 0, \quad |k| = 2.$$

Observe that each term b_j with $|j| = 2$ appears only once in H_2^1, therefore we can isolate it from there to obtain:

$$b_{\overline{k}} = \frac{-1}{a_{1,0}} \left(a_{0,\overline{k}} + \sum_{j+k=\overline{k}} a_{1,j} \cdot H_1^1]^{(k)} + \sum_{j+k=\overline{k}} a_{2,j} \cdot H_2^2]^{(k)} \right).$$

- Collecting terms of order n in $\vec{\xi}$, we get:

$$\begin{array}{lll} \text{for } i = 0 & & P_{0,n} \\ \text{for } i = 1 & P_{1,0}\, H_n^1 + P_{1,1}\, H_{n-1}^1 + \ldots + P_{1,n-1}\, H_1^1 & \\ \text{for } i = 2 & P_{2,0}\, H_n^2 + P_{2,1}\, H_{n-1}^2 + \ldots & \\ \quad \vdots & \vdots & \\ \text{for } i = n & P_{n,0}\, H_n^n & \end{array}$$

Collecting all of them:

$$P_{0,n} \cdot 1 + \underbrace{P_{1,0}}_{a_{1,0}} \underbrace{H_n^1}_{b_j\text{'s}} + \sum_{|j|+|k|=n} P_{1,k} H_j^1 + \sum_{i=2}^{n} \sum_{|j|+|k|=n} P_{i,j} H_k^i = 0, \quad |k| = n.$$

And from this equation, taking each multi-index separately:

$$b_{\tilde{k}} = \frac{-1}{a_{1,0}} \left(a_{0,\tilde{k}} + \sum_{j+k=\tilde{k}} P_{1,k} H_j^1 + \sum_{i=2}^{n} \sum_{j+k=\tilde{k}} P_{i,j} H_k^i \right).$$

Observe that any of the methods (except for Euler-like method) does not require an special section $\mathcal{M} = \{\vec{x} \in \mathbb{R}^n | G(\vec{x}) = 0\}$. If the section is different than $G(\vec{x}) = \vec{x}]_n - C$ but can still be described as a function, we can reproduce all the above computations. The only thing required is to compute the function $G(P(\vec{\xi}))$, after computing the time expansion and before proceeding with the iteration or the recursion.

Numerical results

To test the Poincaré section routines we used the Circular Restricted Three Body Problem equations with the Earth-Moon mass ratio. We take initial conditions at $y = 0$ with a certain velocity on the y component in the negative direction. The surface of section selected is the surface $x = 0$ with $\dot{x} > 0$.

Two different tests have been done in the computation of the Poincaré map, both of them compare the integration using the Jet Transport against a Runge Kutta 7-8. We also compare the three different implementations presented.

Let $\delta t(\vec{\xi})$ be the additional time (positive or negative), that must be added to T, that the initial condition $\vec{x}_0 + \vec{\xi}$ needs to reach the Poincaré section, i.e. $\delta t(\vec{\xi})$ gives the true value of the approximation given by $\tau(\vec{\xi})$. The two values that can be compared are:

- The difference between the value of $\delta t(\vec{\xi})$, computed using the numerical RK78 integration and $\tau(\vec{\xi})$, obtained using one of the previous methods. This difference is shown in the left plot of Figure 2.14.
- The difference between the final $x_{\mathcal{M}}$ value on $\mathcal{M}$ of the numerical RK78 integration and the result of the Jet Transport propagation. These results are shown in the right plot of Figure 2.14.

In both plots the centre of the initial box is at $(-0.43, 0, 0, -0.6)$ and is propagated until it arrives to the manifold $\mathcal{M} = (x, y, \dot{x}, \dot{y}) | x = 0$. Then initial perturbations of the form $(-0.43 + \xi_1, 0, 0, -0.6)$ are considered. The first coordinate of this perturbations is used in the plots as x-axis. In both plots we can see that the mapping of the central point is the one giving best results. This is because the propagation of that point

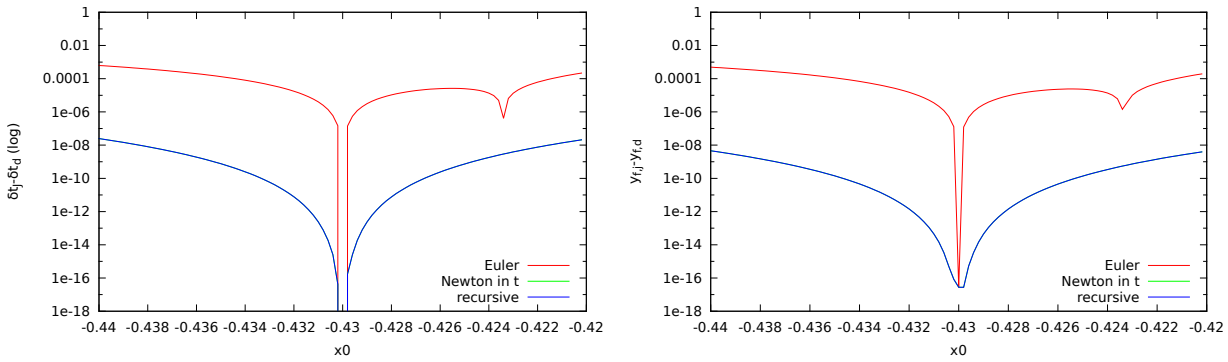


Figure 2.14: The left hand side plot shows the difference between the value of $\delta t(\vec{\xi})$, computed using the numerical RK78 integration and $h(\vec{\xi})$, obtained using Euler's method (red), Newton's method (green) and the recursive method (blue). The right hand side plot displays the difference of final y component for the CR3BP, Earth-Moon case, using the same strategies as for $\delta t(\vec{\xi}) - h(\vec{\xi})$.

only contains errors in the propagation computed by the Jet, i.e. it is correct before applying the procedures. The three methods that we introduced does not modify the mapping through $P_{\mathcal{M}}$ of the central point.

Observe that for both differences, the results obtained using Newton's method for the computation of the Poincaré map and the ones resulting from the recursive formulae are identical. This is because we are obtaining the same polynomial expansions for the functions $\tau(\vec{\xi})$ and $P_{\mathcal{M}}(\vec{\xi})$ in both cases.

Of course, using different orders in the computations using the Jet Transport the results will be different. Figure 2.15 shows the differences in the final y component between the values computed using the numerical RK78 integration and the Poincaré map using the Jet Transport tools to determine the total transfer time with order two (red), four (green) and six (blue). We can see how increasing the order of the gives better results and we are able to enlarge the region where the errors are of the order of the machine error precision.

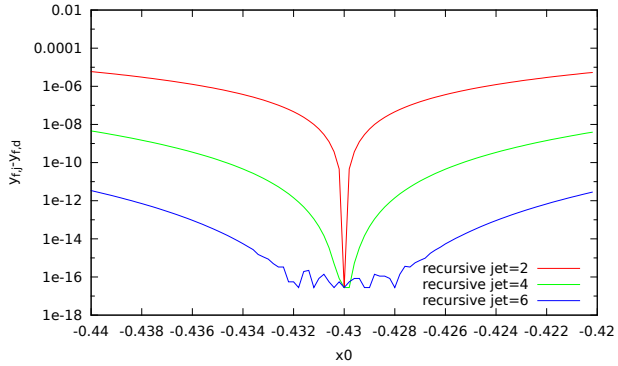


Figure 2.15: Difference of final y component for the CR3BP earth-moon system using a RK78 integration+ Poincaré section and the Jet with the procedure explained in the previous sections. The different plots correspond to different orders in the jet to do all the operations required. Order 2 (red), 4 (green) and 6 (blue).

As a final test, we can explore the effect of using more or less elements of the polynomials $h(\vec{\xi})$ and $P_{\mathcal{M}}(\vec{\xi})$ using the recursion procedure. Figure 2.16 shows the difference in the y component at the Poincaré section between the RK78 integration and the Jet Transport propagation. The different curves show the

differences between the computation of the polynomial $P_{\mathcal{M}}(\vec{\xi})$ computed using different orders of the jet. Observe that, unlike in the previous case, now we are changing the maximum value of the truncation order instead of changing the order up to which we compute the map. We remark that the differences obtained using the recursion up to order n is the best that we can get if our maximum order of computation for all the procedure is n.

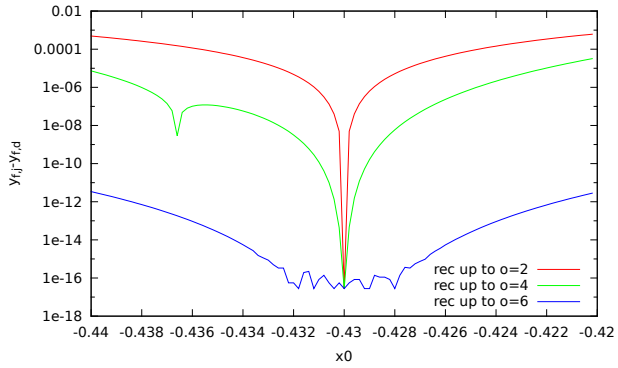


Figure 2.16: Difference of final y component for the CR3BP earth-moon system using a RK78 integration+ Poincaré section and the Jet with the procedure explained in the previous sections. The different plots correspond to different final orders in the computation of the recursive formulas. The polynomial $h(\vec{\xi})$ is computed up to order 2 (red), 4 (green) and 6 (blue).

A final remark on the computational time is worth. We have seen that, from the accuracy point of view, the Poincaré map obtained using Newton's method is identical to the one obtained with the recursive formulae. However, from a CPU time point of view they are not equivalent. Table 2.5 gives the CPU time required by the different methods up to different orders of the Jet Transport and the recursion formulas. In addition, and for the sake of completeness, the table gives also the CPU time required to do the numerical integration. This time should be subtracted since it is the same for all the methods and is not intrinsic to them. We can see how, using the Newton polynomial method takes three orders of magnitude more time than the recursive formulae up to different orders. Furthermore, the time needed to compute the Poincaré map for the recursive formulae is almost negligible compared with the time needed to compute the integration previous to the Poincaré map procedure. Since the recursive methods are so fast, in order to compute the CPU time, the same computation has been done 1000 times. The table shows the CPU time required to do 1000 repetitions as well as the time required to do a single computation of the Poincaré map with jets.

2.5 Domain splitting in flow propagation

One of the main drawbacks of the Jet Transport applied to flow propagations is the fact that the precision decreases as the time of propagation increases. This means that given a certain accuracy, the size of the domain U around the initial condition $\vec{x}_0$, where the Jet Transport propagation can be used within this accuracy, becomes smaller with time.

In a previous section, we have seen how the value for the maximum radius for U can be determined, taking into account the coefficients of the polynomial P have been already computed for a certain propagation time interval.

Ideally, it should be good to know the value of the maximum radius that can be propagated before doing any operation. In this way, knowing the size of the set of initial initial conditions in which we are interested, one can determine the number of neighbourhoods around some of them that will be needed. However, it is not known how to do a priori these kind of computations. The only possible solution is to compute the maximum radius, or some value related to it, using the coefficients of the Jet Transport.

	Jet order 4	Jet order 6	Jet order 6 excluding numerical integration times
Numerical integration	1.48	9.8	
Newton's method	2.09	13.26	44.5s/10it$\rightarrow$ 3.4 s/it
Rec. δt order 2	1.48	9.8	11.104s/1000rep $\rightarrow 1.3 \cdot 10^{-3}$s/rep
Rec. δt order 2	1.48	9.8	14.65s/1000rep $\rightarrow 4.8 \cdot 10^{-3}$s/rep
Rec. δt order 6	1.48	9.8	17.56s/1000rep $\rightarrow 7.7 \cdot 10^{-3}$s/rep

Table 2.5: CPU time, in seconds, required to compute Poincaré maps in the CR3BP using different orders and methods. The times reflected in the last column are repeated 10 times in the Newton method and 1000 times in the recursion methods in order to appreciate the difference of times, both final times are shown in the table first the total time and then the time per repetition.

Hence, if we want to propagate a given set U of the phase space, we fix an initial radius r_0 and an initial condition $\vec{x}_0$, in such way that U is contained in the ball of radius r_0 and centre $\vec{x}_0$. Then we start the propagation; whenever we detect that $P(\vec{\xi})$, with $\vec{\xi} \in U$, is not a good enough approximation of the flow, we proceed with a subdivision strategy, either using two or more polynomials or using two or more regions for the propagation.

The main objective of this section is to explain when and how to do these splittings. The section is divided in two main parts: in the first one we will give a brief explanation of the ideas developed by A. Wittig et al., at Politecnico di Milano, for this problem (see [WDLA$^+$15]). The basic idea is the division and rescaling of polynomials during the integration. The second part describes the procedures developed for this work that are based on the covering of the propagated states by new neighbourhoods.

In both methods we should distinguish two different parts: the first one is related to explain how the division of the neighbourhoods or the polynomials is done, the second one studies when the division should be done.

In both cases, the procedures works schematically as follows. First a certain neighbourhood is propagated until some condition breaks, usually because the precision of the propagation is below some tolerance that we impose. Up to that point the method is a usual Jet Transport flow propagation. When the condition is broken the second part of the algorithm starts, the algorithm or the set of points, depending on the selected strategy, splits. After the division the usual Jet Transport flow propagation is started with the new conditions.

2.5.1 Splitting the polynomial

The first method discussed consists in splitting the polynomial in two new ones whenever some condition is broken. As has already been said, this method was introduced in [WDLA$^+$15].

In this strategy, the initial neighbourhood is parametrised by the square $[-1, 1]^n$. This parametrisation simplifies subsequent computations. At each step of the integration, once the polynomial P is available, the following test is done:

- Given $P(\vec{\xi}) = \sum_k a_k \vec{\xi}^k$, the size of the terms of order i is defined as $S_i = \sum_{|k|=i} |a_k|$.
- For the set I of indices i for which S_i is different from zero, a least squares fit with the exponential function $f(i) = A\ \exp(Bi)$ is used in such a way that $f(i) = S_i$.
- The value of $f(n+1)$ is used to estimate the size S_{n+1} of the truncated order $n+1$ of P. If the difference $S_{n+1} = f(n+1)$ is over a given tolerance then the test fails.

Whenever the test fails, the following algorithm is applied to determine the splitting.

- Look for the more expansive direction e_i, again using an exponential fitting of the coefficients of the polynomial in one variable x_j and taking i as the value which gives more error. The split is only performed in the i direction.

- Do the split of P in two polynomials: $P_\pm = P\left(\xi_1, ..., \xi_{i-1}, \frac{\xi_i}{2} \pm \frac{e_i}{2}, \xi_{i+1}, ..., \xi_n\right)$. Each of the polynomials is defined again in $[-1,1]^n$, but these two boxes are a parametrisation of smaller regions.

After the split, the integration can continue on each one of P_+ and P_-. Taking rectangular boxes has its benefits. The first one is that the boxes will not overlap. In addition, once the splitting strategy is fixed it is straightforward to use. The final result consist in N polynomials. To know which is the propagation of a given point $\vec{x}$ we only need to evaluate the appropriate polynomial at the transformed coordinates in the square $[-1,1]^n$. A possible drawback of the parametrisations by the square $[-1,1]^n$ is the following. In these boxes the corner points are further from the origin than others and it can happen that if the corner points are correctly approximated, there are points outside the box that are approximated by the polynomial with good enough precision but cannot be propagated with the same polynomial.

In the explained procedure, the splitting direction must be one of the directions of the canonical base. It should also be possible to take any direction $\vec{c}$, $\|\vec{c}\| = 1$. Then, the polynomials that must be used in order to expand in that direction are $P_\pm = P\left(\frac{\vec{\xi}}{2} \pm \frac{c}{2}\right)$. This option has the advantage that the worst direction, in the sense of worst approximated, can be taken. However, using the direction $\vec{c}$ the initial region is not completely covered by the two new polynomials $P_\pm$ in $[-1,1]^n$, and in some other parts they overlap each other. This two facts makes unfeasible to consider any direction for the splitting.

2.5.2 Splitting the neighbourhood

The second method splits the domains of integration instead of the polynomial. The main idea is to propagate the initial neighbourhood until a time T when some condition fails. Then a new set of polynomials, which propagate points around $\phi(T; t_0, x_0$, is created according certain rules.

The number of polynomials that we propagate depend on the actual state of the propagation. If the stopping condition fails for some polynomial all the polynomials are used to determine the new set of polynomials.

To detect when the propagation fails, the maximum initial box discussed in section 2.2 is used. At the beginning of the propagation a minimum initial box of size r_0 is fixed. This determines a neighbourhood U_0 that we want to propagate with enough precision. Therefore, we consider that the test fails whenever the size $\xi_{\max}$ of the maximum initial box is below the size r_0.

The next step is to generate new neighbourhoods U_{1_i} in order to cover the propagation of U_0. Two different strategies to determine the covering has been implemented. The first one uses the information that we have in the propagated polynomial to determine where we must put the sets U_{1_i}. The second one uses tracer points to determine the regions that must be cover. In both cases new radius r_{1_i} are determined.

In the next steps of the algorithm there will be, in general, more than one polynomial to integrate. In this situation the maximum initial box test must be checked for each of the polynomials. If one of them breaks the condition, then the splitting procedure is applied to all of them. Observe that now the maximum initial box must be compared with the respective minimum box to propagate r_0.

From now on, we will assume that we are integrating an ODE system using the Jet Transport. At some time T the accuracy test fails, so the maximum value for $\xi_{\max}$ is below the radius r_0 that we want to maintain. At that point we obtain the polynomial $P(\vec{\xi}) = P_{t_0}^T(\vec{\xi}) \approx \phi(T; t_0, \vec{x}_0 + \vec{\xi})$. U will denote the neighbourhood of $\vec{x}_0$ parametrised by $\vec{x}_0 + \vec{\xi}$ with $\|\vec{\xi}\| < r_0$, $V = \phi(T; t_0, U)$ (the image at time T of U computed using the flow), and $\tilde{V} = P(U)$ (the image at time T of U computed using the polynomial P). In general, if U is a ball around $\vec{x}_0$, after some time T its image V is an ellipsoidal (or close to it) neighbourhood.

Divide and conquer

The first first procedure that has been developed uses the information given by $P(\vec{\xi})$ to create the new balls, U_+ and U_-, covering V.

To do that we use the approximated neighbourhood $\tilde{V}$. Using $P(\vec{\xi})$ we can search for the point $\vec{x}_e = P(\vec{\xi}_e)$ such that $\|\vec{v}_e\| = \|P(\vec{\xi}_e) - P(0)\|$ is maximum among the points with $\|\vec{\xi}_e\| = r_0$. This point is obtained refining using Newton's method (and the first derivative of the maximum distance to refine the initial seed) a first guess, obtained in a rough exploration on a grid of points. Note that, since the polynomial $P(\vec{\xi})$ is available, to apply a Newton's method only requires the computation of the derivatives of polynomials, which is done using the formulae of Section 2.2.

In the next step two new neighbourhoods $U_\pm$ are added. They are be located along the direction of the more expanding points, one at each side of the centre of $\tilde{V}$, with their centers at $c_\pm = P(0) \pm \dfrac{\vec{v}_e}{2}$.

The last step consists in the computation of the radius $r_\pm$ of the new initial balls $U_\pm$. Observe that taking $r_\pm = \|\vec{v}_e\|/2$ is not a good option, since using them we are not covering all V, due to that there will be points in the hyper-plane π perpendicular to $\vec{v}_e$, that passes through $P(0)$, that will not be covered by the new neighbourhoods $U_\pm$. Therefore, a search on the half neighbourhood separated by π is done to determine which is the maximum distance $r_\pm$. Again, a grid search is used to determine a good initial guess, that after is refined using Newton's method on the derivative of the distance function. The only difference with respect to the previous step is that now we are restricting the search to one half of the neighbourhood of $\tilde{V}$ determined by π. Figure 2.17 shows a scheme of the different elements that take part of this splitting procedure.

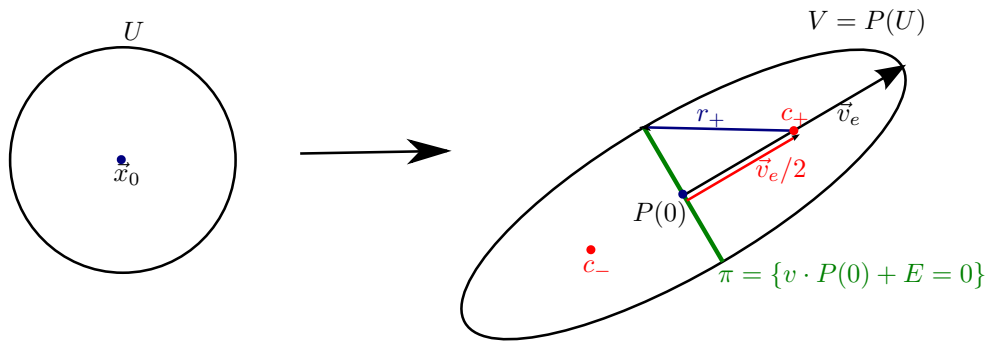


Figure 2.17: Schematic explanation of the splitting region procedure. The propagated ball V is splitted in two new balls with centres along the more expansive direction $\vec{v}_e$ and radius $r_\pm$, in such a way that V is completely covered.

Once the new neighbourhoods $U_\pm$ are computed, they are labelled as U_{n_i}, where n denotes the splitting step in which they are generated and i is an index for its identification. Figure 2.18 shows an example of the successive application of the above algorithm for the simple pendulum when propagating an initial ball located in the lower part of the circulation regime.

In Figure 2.18 shows some of the problems of this splitting procedure. The first and more notorious is that, at the end of the propagation, the covered region is much larger than the one that we need to cover, which is bounded by the green thick line. This is due because at each step we are enlarging the region to be propagated. In principle, this should not be a huge problem, except that it demands a high computational cost, since we are asking to the procedure to propagate bigger boxes than what we really need.

In order to solve this problem we introduce a small variation on the algorithm. In addition to $\vec{v}_e$ we search for $\vec{v}_o$, the maximum expansion direction orthogonal to $\vec{v}_e$. The centre of the new box should be moved closer to $P(0)$ and the radius slightly reduced. Let δ be the distance that we move the centre, according to Figure

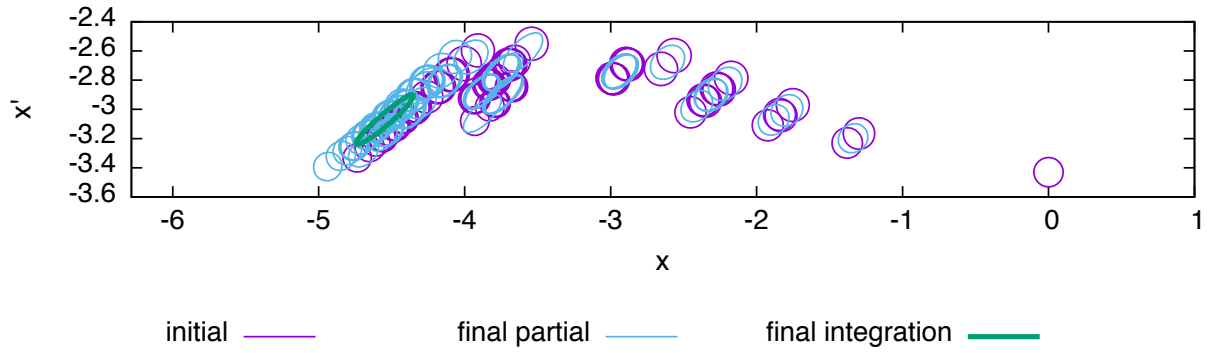


Figure 2.18: Evolution of the splittings, for an initial ball located in the lower part of the circulation region, using the first version of the subdivision algorithm. The initial region is in the right lower part. The purple regions determine the sets U_{n_i}, the blue regions are the propagations of the sets U_{n_i}, $V_{n_i} = P_{n_i}(U_{n_i})$. The green thick line is the true propagation of the initial neighbourhood.

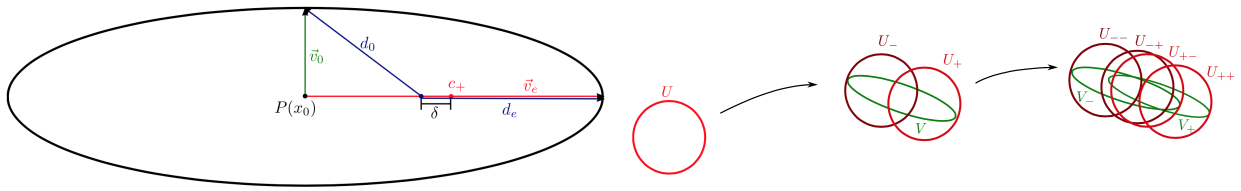


Figure 2.19: Schematic representation of the improved subdivision algorithm. Left: Computation of the δ variation needed to match the distance of maximum expansion and the distance in the normal space. Right: Fusion of neighbourhoods with a large overlapping.

2.19 (left) the final location of the centre must satisfy

$$\frac{\|\vec{v}_e\|}{2} + \delta = \sqrt{\left(\frac{\|\vec{v}_e\|}{2} - \delta\right)^2 + \vec{v}_o^2},$$

from which we get $\delta = \dfrac{\|\vec{v}_o\|^2}{2\|\vec{v}_e\|}$. Therefore, locating the centre of the new neighbourhood $c_\pm = P(0) \pm \left(\dfrac{\|\vec{v}_e\|}{2} - \dfrac{\|\vec{v}_o\|^2}{2\|\vec{v}_e\|}\right)$ and the radius of the neighbourhood $r_\pm = \dfrac{\|\vec{v}_e\|}{2} + \dfrac{\|\vec{v}_o\|^2}{2\|\vec{v}_e\|}$ the new neighbourhood U_{n_i} is the smallest one that covers the corresponding part of V.

Another problem that can be observed in Figure 2.18 is the existence of some neighbourhoods $U_\pm$ that almost completely overlap. This already happens at the second step, because the direction in which the propagation of the initial neighbourhood is expanded is almost the same for each iteration. Figure 2.19 also shows this almost overlapping of the sets U_{+-} and U_{-+}, at the second step of the procedure. To avoid this drawback, a safety factor is added to the method. Once all the new neighbourhoods U_{n_i} are computed, and before proceeding with the next propagation step, the distance between the centres is checked. If there are two centres whose mutual distance is less than 0.2 times their radius, then both neighbourhoods are joined. The centre of the new set U_{n_i} is located at the middle point of the two previous ones and the radius enlarged by one half of the distance between the two centres.

Using this last procedure we are able to propagate the neighbourhoods with the desired precision. How-

ever, there is still some overlapping of the sets U_{n_i} when their centres are close but not close enough to be merged by the improvement. The next algorithm presents an alternative way to place the neighbourhoods U_{n_i}. It uses the information given by some tracers, which are defined as points $T_{0,i}$ in U_0 that are propagated up to the splitting time to determine the region V to be covered by the new sets U_{n_i}.

The explorers: send tracers to control the space

Consider an initial set of points $\mathcal{T}_0 = \{\vec{t}_{0,1}, \ldots, \vec{t}_{0,m}\}$ on U_0, and propagate them up to the splitting time T using the polynomial $P(\vec{\xi})$ that has already have computed. this are the points that will be to determine the new sets U_{n_i} that allow the continuation of the propagation without loosing precision.

Let us consider first the algorithm in the 2-dimensional case. In this situation, a good option for the selection of the points of $\mathcal{T}_0$ is to take them in the boundary of U_0, this is: $\vec{t}_{0,i} = \vec{x}_0 + (r_0 \cos(2\pi i/m), r_0 \sin(2\pi i/m))$, together with the central point $\vec{t}_{0,0} = x_0$. Next, propagate $\vec{x}_0$, using the Jet Transport, until the splitting condition is reached. In that way we obtain the polynomial $P^T_{\mathcal{T}_0}(\vec{\xi})$.

Using $P^T_{\mathcal{T}_{n-1}}(\vec{\xi})$, determine the set of tracers at T: $\vec{t}_{n,i} = P^T_{\mathcal{T}_{n-1}}(\vec{t}_{n-1,i})$. These are the points that we want to cover with the sets U_{n_i}. The algorithm that follows is used to determine the U_{n_i} that will be propagated in the next integration step. The idea is to use a grid of boxes of a fixed radius that will be used to determine the position of the sets U_{n_i}. A set U_{n_i} will be located at a given box if it contains a tracer. Observe that inside the grid each box has a multi-index k associated. The box with multi-index k is that one located in the k_i-th position in the i-th direction of the grid.

Algorithm 1. At the j-th splitting step,

1. Fix a value r for the radius of the sets U_{j_i} and compute the length c of the side of a square (or in general a hyper-cube) such that r is the length of the diagonal, $c = \sqrt{\dfrac{4r^2}{N}}$, where N is the dimension of the system.
2. For each component of the tracers $\vec{t}_{j,i}$, compute the minimum $a_\iota = \min_{0\le\iota\le m} \vec{t}_{j,i}]_\iota$ and the maximum $b_\iota = \max_{0\le\iota\le m} \vec{t}_{j,i}]_\iota$. These two values will be used to determine the extremes of the grid.
3. Compute the grid size $\ell_\iota = \left(\left\lfloor \dfrac{b_\iota - a_\iota}{c} \right\rfloor + 1 \right) c$.
4. Fix an N-dimensional grid. Each side will have $\left\lfloor \dfrac{b_\iota - a_\iota}{c} \right\rfloor + 1$ boxes, and will start at $\tilde{a}_\iota = a_\iota - \dfrac{\ell_\iota - b_\iota + a_\iota}{2}$.
5. If a box with multi-index k contains a tracer $\vec{t}_{j,i}$, locate in it a new set $U_{j,i}$. The coordinates of the centre of the neighbourhood $U_{j,i}$ are $a_i + (k_i + 0.5) \cdot c$ and the radius is r.
6. If a given box contains two tracers, initialise only one polynomial.

Observe that the grid size defined in step 3 is bigger than the distance between the maximum and the minimum in each direction. This increases the region where new sets are selected and, in this way, if the set of tracers does not arrive to the point which has more expansion there are still chances to capture it.

Figure 2.20 shows an example of the elements introduced in the algorithm. It also gives an example of the steps of the procedure for a pendulum propagation in the lower circulating zone. The purple curve is the "true" propagation by the flow of the set U_0, the tracers are propagated using the polynomial P and marked with green crosses. The grid is represented in blue, and the final neighbourhoods to be propagated in yellow. The bottom right image shows the result of the propagation of the sets $U_{1,i}$.

The left hand side plots of Figure 2.21 show the result of the second, third and fourth step of the splitting procedure using tracers. The purple curve is the propagation of the boundary of the set U_0, up to the time

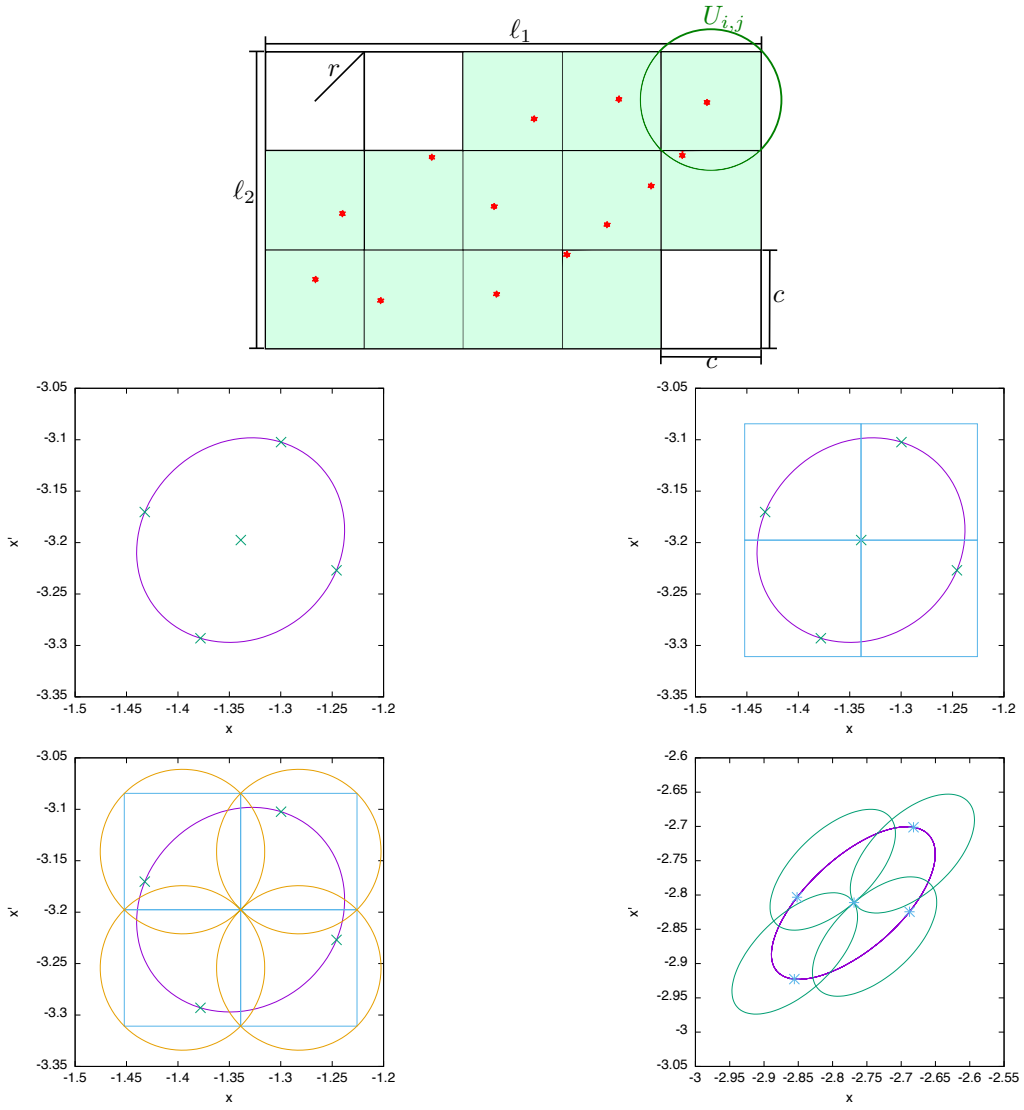


Figure 2.20: The top figure displays the different elements that take part in the selection of the neighbourhoods U_{n_i}. The red dots are the position of the tracers and the light green boxes the ones that will contain a new neighbourhood $U_{i,j}$. The four bottom figures show, from left to right and up to bottom, a sequence of the neighbourhood selection for an initial condition in the lower circulating zone of a pendulum. The purple curve is the "true" propagation by the flow of the set U_0 using Taylor's method, the tracers are propagated using the polynomial P and marked with green crosses. On them we set the grid (in blue) to select the new neighbourhoods U_1 that will be propagated (yellow). The bottom right image shows the result of the propagation of the sets $U_{1,i}$, the purple ellipse is again the propagation of the boundary of U_0 of the box using a Taylor integration method up to the time on which the splitting condition holds. The blue crosses are the propagation of the tracers and the green small ellipsoids are the propagation of the boundaries of the sets $U_{1,i}$.

when the splitting condition is activated. The tracers, propagated using the Jet Transport polynomial P, are marked with green crosses, the grid determined at each step is in blue and the new neighbourhoods U_{n_i} are shown in yellow. The last plot corresponds to the last step of the integration, when the final desired time has been reached. There, we can see in purple the result of the "true" integration using the flow and, in green, the boundary of the sets V_{3_i} at the final time. In this last plot it can be seen that the green sets do not cover completely the purple one. This happens because during the previous splitting step not all the boxes, which contain a part of the propagation of U_0, are selected, due to the expansion of the set U_0. The set U_0 expands with time, therefore, the tracers containing it will also expand and separate each other.

To avoid big distances between the tracers, new ones are added during the splitting procedure. Except $\vec{t}_{0_0}$, that is at the centre of U_0, all the other tracers are ordered at the boundary of U_0. Therefore, if at the n-th splitting iterate the distance between two consecutive tracers is grater than d_{tol}, $\|\vec{t}_{n_i} - \vec{t}_{n_{i+1}}\| > d_{tol}$, then a new tracer is added at the boundary between them. To add a tracer, first we compute the middle point of $\vec{t}_{0,i}$ and then $\vec{t}_{0,i+1}$, that must be normalised to be at the boundary of U_0. The new tracer is added at

$$\tilde{\vec{t}}_{0,i+1} = \vec{x}_0 + r_0 \frac{\vec{t}_{0_i} - \vec{t}_{0_{i+1}} - \vec{x}_0}{\|\vec{t}_{0_i} - \vec{t}_{0_{i+1}} - \vec{x}_0\|}.$$

Once a tracer is added in the right place, observe that the tracers with an index grater than i will change their index in one unit. Finally, the new tracer is propagated to the actual time in order to start the splitting procedure.

The right hand side plots of Figure 2.21 show the second and third steps of the splitting procedure, as well as the result at the final time, using the improved algorithm that adds tracers to the set. Observe that now the final propagation of U_0 is correctly covered by the propagation of the U_{n_i}. However, the covering is not done with the minimum number of neighbourhoods. It is clear that the last step can be done in five or even four sets instead of the seven that the procedure is currently using. To overcome this drawback, is convenient to put the grid along the maximum expansion direction, in this way there will be a direction with an elongated grid an another, in the orthogonal direction, with a narrow grid. This can be done according to the following:

Algorithm 2 (Adapted direction to determine splits using tracers)**.** Once the propagation is stopped by the splitting condition

1. Find the most expanding direction $\vec{v}_e$.
2. Use a Gram-Smidth method to compute an orthonormal basis which contains $\vec{v}_e$ as one of the vectors. Let C be the matrix of the change of basis.
3. Transform the position of the tracers to the new base: $\vec{\bar{t}}_{n_i} = C\vec{t}_{n_i}$.
4. Apply Algorithm 1 to determine the centres $\vec{\bar{c}}_i$ of the neighbourhoods that will be propagated.
5. Transform the position of the centres back to the initial coordinate frame: $\vec{c}_i = C^*\vec{\bar{c}}_i$.
6. The radius of the neighbourhoods do not change with this algorithm, since the transformation used is orthonormal.

Figure 2.22 shows the results of the above adapted direction algorithm. The final result show that the number of neighbourhoods has been reduced from the seven that the unmodified algorithm had to only four.

Figure 2.23 shows the precision of the results obtained for the simple pendulum with initial condition $\vec{x}_0$ in the lower circulation zone. One can see that the errors are of the order of 10^{-8}. Taking into account that the precision required for the splitting condition is to maintain the precision up to 10^{-6}, the errors obtained are reasonable. The right hand side plot of Figure 2.23 shows the error of the propagations as a function of the final position of the points. One can see that the points that are closer to the boundary have higher errors than the ones close to the centre. Therefore, it can be useful to add an additional set U_{n_i} to the propagation at the centre surrounding the image of $\vec{x}_0$, if these are the points for which more accuracy is required.

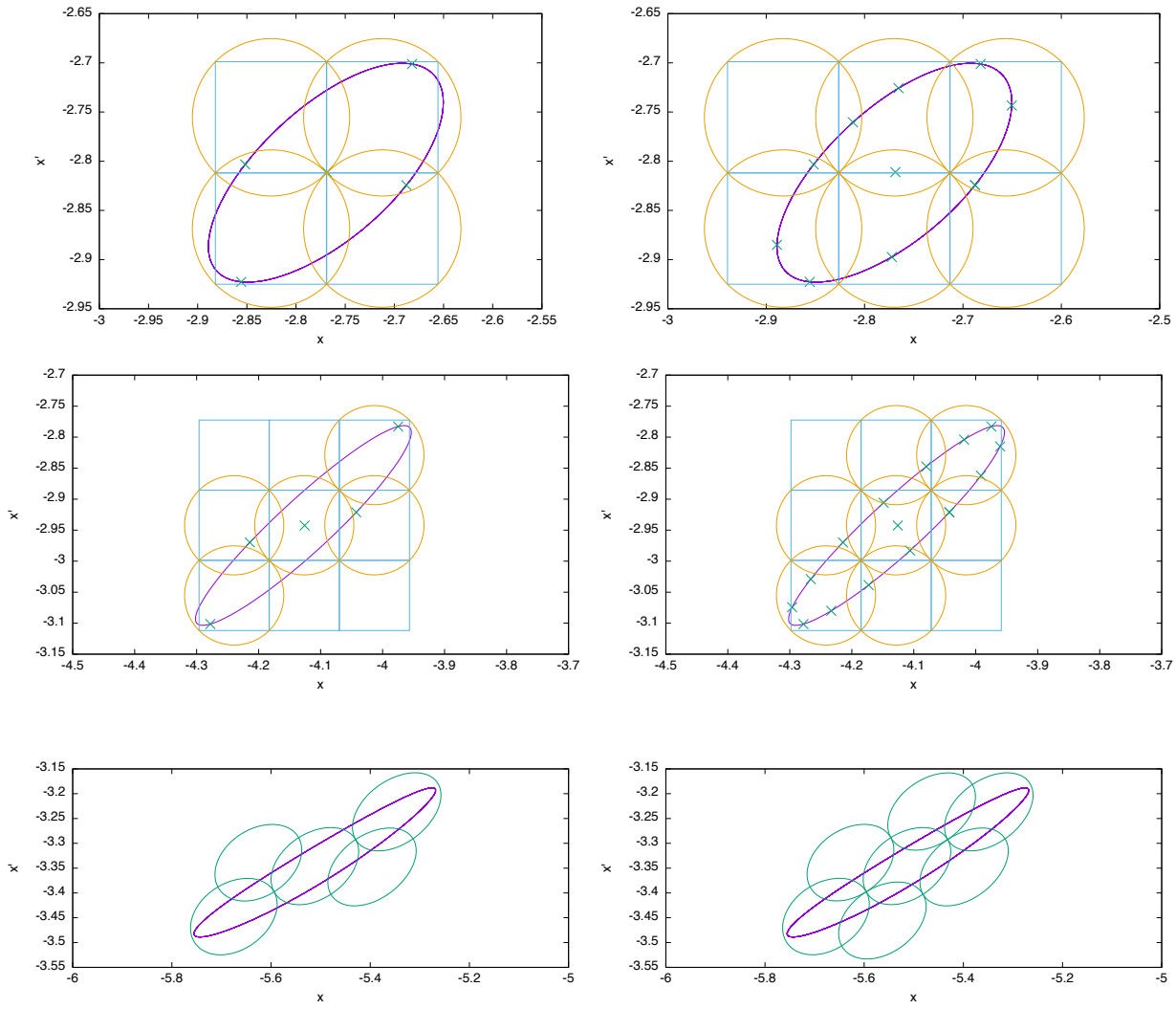


Figure 2.21: The the first two plots on left hand side show the results of the first and second splitting iterates, using the basic algorithm. The purple curve is the true propagation of the initial neighbourhood U_0, the green crosses are the propagation of the tracers, the blue grid is the grid used to determine the new sets U_{n_i}, and the yellow circles the sets U_{n_i} that we will propagate. The bottom plot on the left shows the result at the final integration time. In purple is represented the final neighbourhood, and in green the neighbourhoods propagated in the last step. The right hand side plots correspond to the same computations as in the left but adding tracers in the procedure when the initial ones are separated more than a fixed distance.

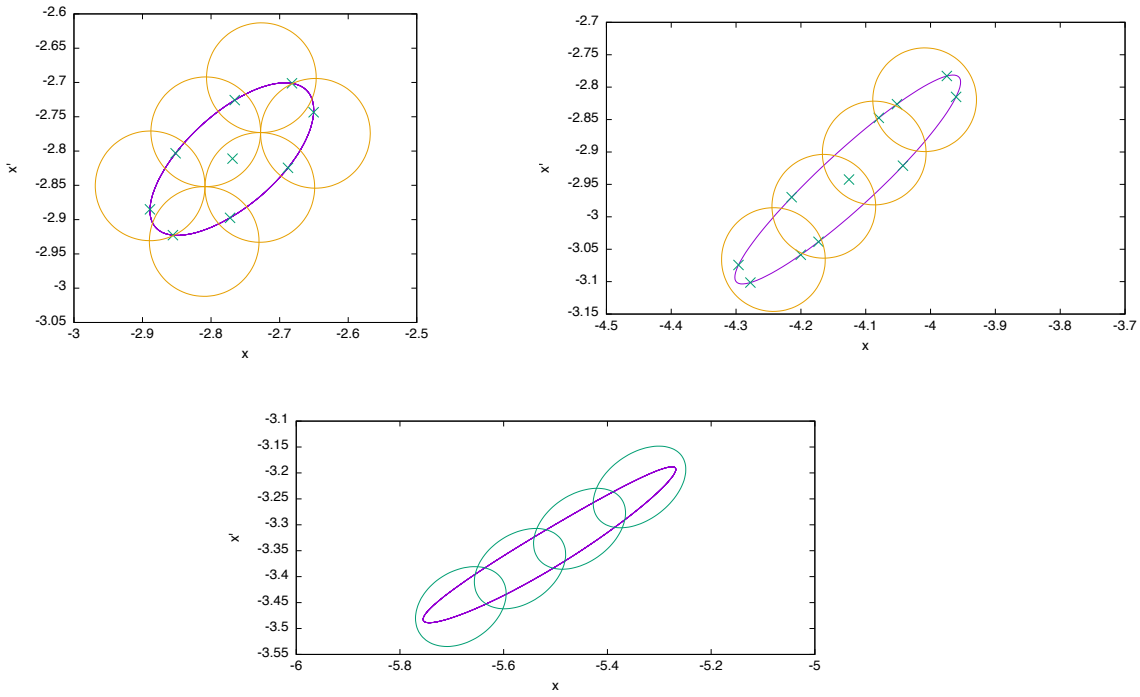


Figure 2.22: The two top pictures show th results of the first and second splitting iterates using the modified algorithm with search of new sets U_n in the adapted base. The purple curve is the true propagation of the initial neighbourhood U_0, the green crosses are the propagation of the tracers and the yellow circles the sets U_{n_i} that we will propagate. The bottom plot displays the result at the final integration time and can be compared with the one in Figure 2.21. The purple curve is the final neighbourhood and in green curves are the neighbourhoods propagated in the last step.

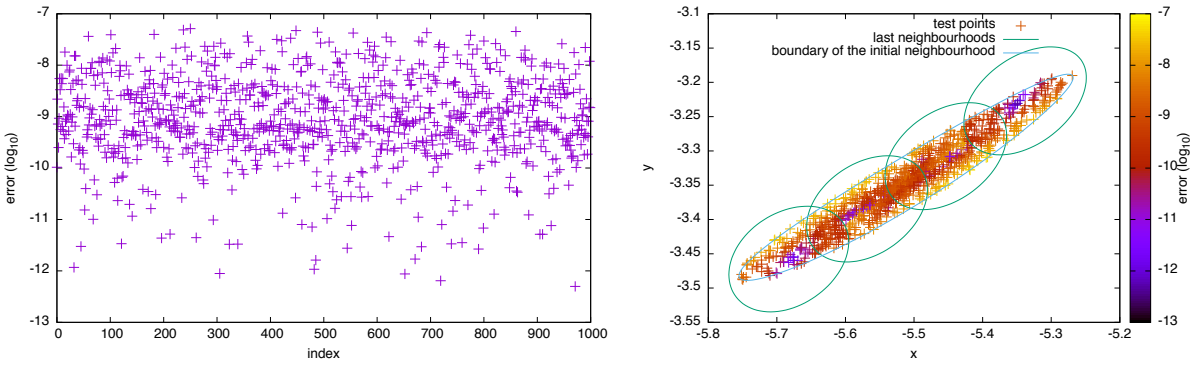


Figure 2.23: The left hand side plot displays the errors of the propagation using the splitting algorithm, for initial conditions around x_0 in the lower circulation zone. The initial conditions are taken randomly around this point. The index on the x-axis is an index assigned to the 1000 initial conditions used. The y-axis gives the difference between the propagation using a RK-78 numerical integration and the polynomials obtained using the algorithm with adapted search grids. The right hand side plot of Figure 2.23 shows the error of the propagations as a function of the final position of the points. The green ellipses are the final propagation of the neighbourhoods obtained in the last step.

The algorithm can be implemented in higher dimensions. The only thing that requires some attention is how to maintain the tracers. In the two-dimensional case, it is easy to maintain them checking the distance between two consecutive ones. In higher dimensions there is no trivial order in the boundary of an n-sphere.

In the high dimensional case, each tracer $\vec{t}_{0_i}$ is identified by an index, as in the 2-dimensional case, but has also associated an array with the indices the other tracers to which $\vec{t}_{0_i}$ is connected. The length of the array is 2^{n-1}, where n is the dimension of the system. Since the tracers also generate hyper-polyhedra, is is possible to add to the previous index an array of all the polyhedra with the tracers that are at its vertices. Two tracers are in the same polyhedron if they are connected by an edge.

It remains to study how to determine when a tracer must be added and how to do it. Observe that in the 2-dimensional case there is only one dimension to measure how far two tracers are, because each tracer only has one direction to go to the next one. In the high dimensional case there are two different ways for adding a tracer:

- Compute the distance between all the connected tracers. If the distance between two of them is higher than a given tolerance d_{tol}, then a tracer is added.
- Compute the n-volume generated by each hyper-polyhedron. If it is higher than a certain value v_{tol} then a new tracer is added.

Observe that the volume criteria has a problem: it may happen that all the vertices of a polyhedron, except one, are close, in this situation the volume of the polyhedron will be small but there will be a tracer far away. In the 3D case the polyhedron will be a triangle, we can put a vertex of the triangle as far as we want an have the other two in such a way that the surface (the 2-dimensional volume) is small. Therefore, there should be a new tracer in between. However, since the volume is small it is not added. Because of that this strategy has been discarded.

In the two-dimensional case, when a new tracer is added, we normalise the baricenter of the two distant tracers is normalised. Following this idea, there are now two options for adding a new tracer:

- Normalise the baricenter of the vertices of the polyhedron with distant tracers.
- Normalise the baricenter of the distant tracers. long distance.

Figure 2.24 shows the two different ways of adding a new tracer. There it can be seen that if we use the first option, adding at the baricenter of the polyhedron, the long distance (in red) will remain, and therefore the procedure will be repeated indefinitely without affecting the two tracers. Because of this, the first option has been discarded. The second option is the one that reduces the length of the detected too long distance between tracers. In its implementation we must take into account that the array of the other tracers related to that one has also been modified accordingly. Depending on how the checks for the addition of the tracers are done it may happen that some new tracers are duplicated. In that case they must be eliminated in order to do not duplicate information.

Figure 2.25 shows the errors of a propagation using the splitting procedure with respect to a RK-78 numerical integration of Kepler's problem. For the computations, 1000 random initial conditions have been used. The left plot gives the error between the result of the RK-78 numerical integration and the splitting domain propagation, while the right hand side picture displays the error in the energy. In both cases the errors are slightly below the error of the splitting condition, 10^{-6}.

2.6 Inversion of polynomials

In some applications, it is interesting to know not only the Taylor expansion of a function but also the expansion of its inverse. The objective of this section is to determine the jet of the function f^{-1}, assuming that the jet of f, given by $\sum_k a_k \vec{\xi}^k$, is known.

Let the Taylor expansion of f be given by: $P(\vec{\xi}) = \sum_k a_k \vec{\xi}^k \sim f(\vec{x}_0 + \vec{\xi})$, with all the coefficients a_k known, and the expansion of f^{-1} be: $Q(\vec{\xi}) = \sum b_k \vec{\xi}^k \sim f^{-1}(a_0 + \vec{\xi})$, where the coefficients are the ones that must be determined. Observe that if $f(\vec{x}_0) = a_0$, then $f^{-1}(a_0) = \vec{x}_0$, therefore, $b_0 = \vec{x}_0$.

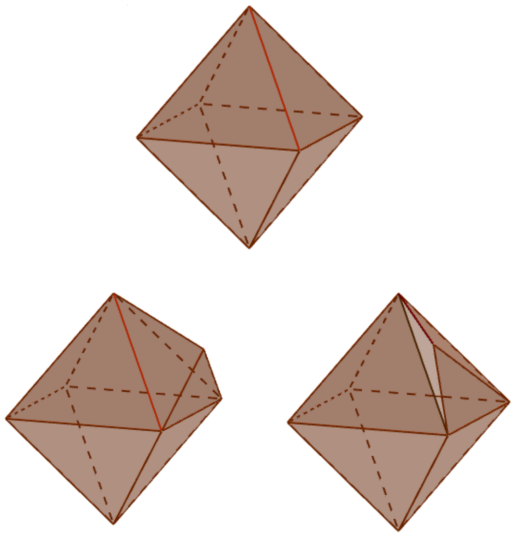


Figure 2.24: In the top figure, initial situation of the tracers (vertices of the polyhedron) in a 3-dimensional system. The red line between the two traces is assumed to be too long. The two bottom figures show the two possible ways to add a new tracer. The left one shows how it is added using the face of the polyhedron to determine the new position: first the baricenter of all the vertices is computed and then normalised to be at a distance r_0. The right one shows how to add the tracer using only the two tracers that are next to it. Again their baricenter is computed and then normalised to be at the correct distance.

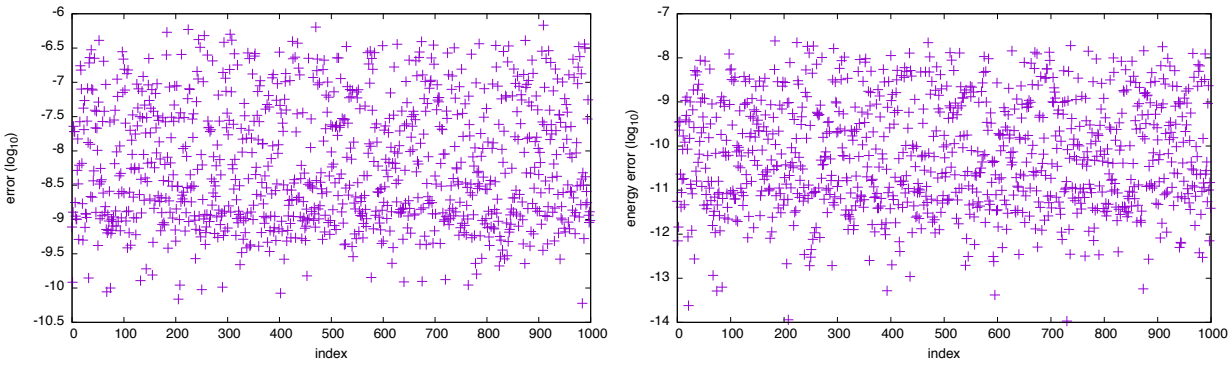


Figure 2.25: Test of an integration of the Kepler's problem to check the precision of the splitting sets propagation. Left: difference between the splitting propagation and the RK-78 numerical integration. Right: difference between the initial and final Keplerian energy of the orbits.

It is useful to assume that the independent term of P, a_0, is zero. This can be done subtracting a_0 to the value of the function f. Since we are subtracting a constant, all the terms of the Taylor series do not change. The polynomial P gives an estimate of the difference between $f(\vec{x}_0) = 0$ and $f(\vec{x}_0 + \vec{\xi})$. Then, the polynomial Q approximates $f^{-1}(\vec{x}) - \vec{x}_0$. In such way it verifies that $0 = Q(P(0)) = Q(0)$, where the first equality is due to the fact that Q and P are functional inverses and the second one to the assumption $P(0) = 0$.

To compute the unknown coefficients b_k two different strategies have been implemented.

- The first strategy solves the non-linear equation $P(Q(\vec{\xi})) - \vec{\xi} = 0$ iteratively, considering $Q(\vec{\xi})$ as a polynomial variable. This has been done in two different ways: using a fixed point procedure, in which at each iteration all the coefficients of Q of a certain order are determined, and using Newton's method, with which at each iteration the number of orders for which all the coefficients are computed is multiplied by 2.

- The second strategy uses of recursive formulae to determine the coefficients of b_k, requiring to the composition of both polynomials to be the identity: $P(Q(\vec{\xi})) = \vec{\xi}$. The formulae for the computation of the b_l of order l, use the coefficients of P and the already known coefficients b_k with $|k| < l$. The b_l are obtained as the solution of a linear system of equations.

2.6.1 Iterative inversion of polynomials

The first strategy for the computation of the inverse polynomial, of a given polynomial $P(\vec{\xi})$, is based in the use of the iterative procedures used to solve non-linear equations, such as the fixed point method or Newton's method. In the two cases, a general formula is given to be applied iteratively. At each iteration a new polynomial $Q_n(\vec{\xi})$, that approximates with higher order the inverse function, is obtained.

The fixed point method

A fixed point method to solve $f(\vec{x}) = 0$ requires the construction of a function g that verifies $g(\vec{x}) = \vec{x}$ when $f(\vec{x}) = 0$. If there exists $\vec{x}_0$ such that $f(\vec{x}_0) = 0$, then there exists a fixed point of $g(\vec{x})$ at $\vec{x}_0$.

The equation that we want to solve is:

$$P(Q(\vec{\xi})) = \vec{\xi},$$

where P and Q are polynomials with the coefficients of P known. We will make use of the fact that $Q(\vec{\xi})$ can be considered as a jet.

Writing the full expression of P, the previous formula becomes:

$$\vec{a}_0 + \sum_{|k|=1} a_k Q(\vec{\xi})^k + \sum_{|k|>1} a_k Q^k(\vec{\xi}) = \vec{\xi}.$$

The second term of this equation, that involves the order one terms, can be written in matrix form as $A_1 Q(\vec{\xi})$, since all the multi-indices k involved in it are related to just one of the polynomials. Now, isolating the Q term that appears in this second summand, we get:

$$Q(\vec{\xi}) = A_1^{-1} \left(-\vec{a}_0 - \sum_{|k|>1} a_k Q^k(\vec{\xi}) + \vec{\xi} \right).$$

Finally, taking into account the condition $P(0) = a_0 = 0$, the resulting iteration formula is:

$$Q_{n+1}(\vec{\xi}) = A_1^{-1} \left(\vec{\xi} - \sum_{|k|>1} a_k Q^k(\vec{\xi}) \right).$$

The only thing that remains is to decide which is the initial seed for the procedure. It has to be close to the final condition. Since the objects of interest are polynomials, a good first guess is the constant part of Q, i.e. $Q_0(\vec{\xi}) = 0$. With this initial seed, at each iteration step the order of the coefficients that are determined increases in one unit.

Newton's method

Newton's method looks for the solution of the equation $f(\vec{x}) = 0$. To do so, given an initial condition $\vec{x}_0$, the following iteration formula is used:

$$\vec{x}_{n+1} = \vec{x}_n - (Df(\vec{x}_n))^{-1} f(\vec{x}_n).$$

In our case, the equation to be solved is $F(Q(\vec{\xi})) = P(Q(\vec{\xi})) - \vec{\xi} = 0$ and the unknown $Q(\vec{\xi})$. Accordingly, the iteration formula is:

$$Q_{n+1}(\vec{\xi}) = Q_n(\vec{\xi}) - (DF(Q_n(\vec{\xi})))^{-1} F(Q_n(\vec{\xi})),$$

and, since $\vec{\xi}$ is constant in the polynomial algebra setting, $DF(Q(\vec{\xi})) = DP(Q(\vec{\xi}))$, therefore:

$$Q_{n+1}(\vec{\xi}) = Q_n(\vec{\xi}) - \left(DP(Q_n(\vec{\xi}))\right)^{-1} \left(P(Q(\vec{\xi})) - \vec{\xi}\right),$$

that can be used together with the initial condition $Q_0(\vec{\xi}) = 0$, i.e. the constant polynomial which solves $P(x_0) = 0$.

Efficiency of the methods

In this section we will prove that the fixed point algorithm computes one additional order of the inverse polynomial at each step, while Newton's method has quadratic convergence, this is, the number of orders computed at each step is multiplied by 2.

In order to proceed with the proofs let us define first the inverse of order n of a certain polynomial.

Definition 1. Given a polynomial $P(\vec{\xi})$, a polynomial $Q(\vec{\xi})$ is said to be an inverse of order n of $P(\vec{\xi})$ if $P(Q(\vec{\xi})) = \vec{\xi} + o(\vec{\xi}^n)$.

Observe that, with this definition, the polynomial $Q_0(\vec{\xi}) = 0$ is an inverse of the polynomial $P(\vec{\xi})$ of order 1.

Proposition 2. *Given a polynomial $P(\vec{\xi}) = \sum_{|k|<n} a_k \vec{\xi}^k$ and an initial polynomial $Q_0(\vec{\xi}) = 0$ then,*

- *The iterate $Q_n(\vec{\xi})$ of the fixed point method is an inverse of order $n+1$ of $P(\vec{\xi})$.*
- *The iterate $Q_n(\vec{\xi})$ of the Newton's method is an inverse of order 2^n of $P(\vec{\xi})$.*

Proof. Let us start with the fixed point method. We will proceed by induction. Assuming that $Q_0(\vec{\xi}) = 0$ then $Q_1(\vec{\xi}) = A_1^{-1}\vec{\xi}$, therefore:

$$P(Q_1(\vec{\xi})) = P(A_1^{-1}\vec{\xi}) = A_1 A_1^{-1}\vec{\xi} + \sum_{|k|\geq 2} a_k \left(A_1^{-1}\vec{\xi}\right)^k = \vec{\xi} + o(\vec{\xi}^2),$$

which means that $Q_1(\vec{\xi})$ is an inverse of $P(\vec{\xi})$ of order 2.

Now assume that $Q_n(\vec{\xi})$ is an inverse of order $n+1$, this implies that:

$$P(Q_n(\vec{\xi})) = A_1 Q_n(\vec{\xi}) + \sum_{|k|\geq 2} a_k Q_n^k(\vec{\xi}) = \vec{\xi} + o(\vec{\xi}^{n+1}).$$

From this equation we get that $\vec{\xi} - \sum_{|k|\geq 2} a_k Q_n^k(\vec{\xi}) = A_1 Q_n + o(\vec{\xi}^{n+1})$. Applying the iterative method we

obtain: $Q_{n+1}(\vec{\xi}) = A_1^{-1}\left(\vec{\xi} - \sum_{k\geq 2} a_k Q_n^k(\vec{\xi})\right)$. Finally, doing the composition with $P(\vec{\xi})$ we get:

$$\begin{aligned}
P(Q_{n+1}(\vec{\xi})) &= A_1 A_1^{-1}\left(\vec{\xi} - \sum_{|k|\geq 2} a_k Q_n^k(\vec{\xi})\right) + \sum_{|k|\geq 2} a_k \left(A_1^{-1}\left(\vec{\xi} - \sum_{\tilde{k}\geq 2} a_{\tilde{k}} Q_n^{\tilde{k}}(\vec{\xi})\right)\right)^k \\
&= \vec{\xi} - \sum_{|k|\geq 2} a_k Q_n^k(\vec{\xi}) + \sum_{|k|\geq 2} a_k \left(A_1^{-1}\left(A_1 Q_n(\vec{\xi}) + o(\vec{\xi}^{n+1})\right)\right)^k \\
&= \vec{\xi} - \sum_{|k|\geq 2} a_k Q_n^k(\vec{\xi}) + \sum_{|k|\geq 2} a_k \left(Q_n(\vec{\xi}) + o(\vec{\xi}^{n+1})\right)^k \\
&= \vec{\xi} - \sum_{|k|\geq 2} a_k Q_n^k(\vec{\xi}) + \sum_{|k|\geq 2} a_k Q_n^k(\vec{\xi}) + o(\vec{\xi}^{n+2}) = \vec{\xi} + o(\vec{\xi}^{n+2}).
\end{aligned}$$

Observe that $\left(Q_n(\vec{\xi}) + o(\vec{\xi}^{n+1})\right)^k = Q_n^k + o(\vec{\xi}^{n+|k|})$. Indeed,

$$\left(Q_n(\vec{\xi}) + o(\vec{\xi}^{n+1})\right)^k = Q_n^k(\vec{\xi}) + \sum_{i=0}^{l} \underbrace{Q_n^{k-e_i}(\vec{\xi})}_{o(\vec{\xi}^{|k|-1})} \cdot o(\vec{\xi}^{n+1}) + o(\vec{\xi}^{2n+2}).$$

Since the smallest k that we have is of order 2, what we have finally obtained is that the final order is $n+2$.

Observe that the fact that $Q(0) = 0$ plays an important role in the computation of the new step. If this is not true we can not warranty that $\left(Q_n(\vec{\xi}) + o(\vec{\xi}^{n+1})\right)^k = Q_n^k + o(\vec{\xi}^{n+|k|})$.

Consider Newton's method. The objective is now to prove that $Q_n(\vec{\xi})$ is an inverse of order 2^n of $P(\vec{\xi})$. This statement is equivalent to proof, also by induction, that $Q_n(\vec{\xi})$ is a zero of the function $F(Q(\vec{\xi})) = P(Q(\vec{\xi})) - \vec{\xi}$. Assuming that $Q_0(\vec{\xi}) = b_0$ we get that:

$$Q_1(\vec{\xi}) = b_0 - (DP(b_0))^{-1}\left(P(b_0) - \vec{\xi}\right) = b_0 + A_1^{-1}\vec{\xi}.$$

Then, $Q_1(\vec{\xi})$ is a zero of order 2 of F:

$$\begin{aligned}
F(Q_1(\vec{\xi}) &= F(b_0 + A_1^{-1}\vec{\xi}) = P(b_0 + A_1^{-1}\vec{\xi}) - \vec{\xi} \\
&= P(b_0) + DP(b_o)DP^{-1}(b_0)\vec{\xi} + D^2P(b_0)DP^{-1}(b_0)\vec{\xi}^2 + \text{ h.o.t. } - \vec{\xi} \\
&= D^2P(b_0)\left(DP^{-1}(b_0)\vec{\xi}\right)^2 + \text{ h.o.t. } = o(\vec{\xi}^2).
\end{aligned}$$

Assume now that $Q_n(\vec{\xi})$ is an inverse of order 2^n, or, equivalently, that $Q_n(\vec{\xi})$ is a solution of order 2^n of F, i.e. $F(Q(\vec{\xi})) = o(\vec{\xi}^{2^n})$. Let us see that: $Q_{n+1}(\vec{\xi}) = Q_n(\vec{\xi}) - DP(Q_n(\vec{\xi}))^{-1}\left(P(Q_n(\vec{\xi})) - \vec{\xi}\right)$ is a zero of

order 2^{n+1} of F. Expanding F by its Taylor expansion at $Q_n(\vec{\xi})$ and applying it to $Q_{n+1}(\vec{\xi})$ we get:

$$\begin{aligned}
F(Q_{n+1}(\vec{\xi})) &= F\big(Q_n(\vec{\xi}) - DP(Q_n(\vec{\xi}))^{-1}\left(P(Q_n(\vec{\xi})) - \vec{\xi}\right)\big) \\
&= F(Q_n(\vec{\xi})) - DF(Q_n(\vec{\xi}))\left(\underbrace{DP^{-1}(Q_n(\vec{\xi}))}_{DF(Q_n(\vec{\xi}))^{-1}}\underbrace{\left(P(Q_n(\vec{\xi})) - \vec{\xi}\right)}_{F(Q_n(\vec{\xi}))}\right) \\
&\quad + \frac{1}{2}D^2F(Q_n(\vec{\xi}))\left(DP^{-1}(Q_n(\vec{\xi}))\left(P(Q_n(\vec{\xi})) - \vec{\xi}\right)\right)^2 + \dots \\
&= \frac{1}{2}D^2F(Q_n(\vec{\xi}))\left(DF(Q_n(\vec{\xi}))^{-1}\left(\underbrace{F(Q_n(\vec{\xi}))}_{o(\vec{\xi}^{2^n})}\right)\right)^2 + \dots \\
&= \underbrace{\frac{1}{2}D^2F(Q_n(\vec{\xi}))}_{o(1)}\underbrace{\left(DF(Q_n(\vec{\xi}))^{-1}\right)^2}_{o(1)}\underbrace{\left(F(Q_n(\vec{\xi}))\right)^2}_{o(\vec{\xi}^{2^{n+1}})} + \dots = o(\vec{\xi}^{2^{n+1}}).
\end{aligned}$$

Observe that $DF(Q_n(\vec{\xi}))$ coincides with $DP(Q_n(\vec{\xi}))$ because $\vec{\xi}$ is a constant in the truncated polynomial algebra variables. Note also that the derivatives of F applied to Q_n are of order 1. When we differentiate F we obtain the matrix A_1 from $P(\vec{\xi})$ as independent term, that matrix is not necessary cancelled when we apply $Q_n(\vec{\xi})$. □

Tables 2.6 and 2.7 give the results of a numerical test that shows which orders are correctly computed at each iteration. This is done comparing the iterate Q_n with the polynomial Q obtained at the end of the procedure. At each order, the table shows the average of the decimal logarithm of the errors of all coefficients of the vectorial polynomial in the given order. The polynomial used to check that result is the same used to check the polynomial inversion in the next subsection. The first coefficients of the polynomial and the inverse computed can be found in Table 2.8.

order	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
iterate 0	-18	-18	-12	-12	-9.8	-11	-11	-9.3	-7.7	-8.1	-8	-9.4	-9.8	-12	-13	-12
iterate 1	-18	-18	-18	-9.8	-8	-11	-7.5	-8.8	-11	-8.2	-9.4	-11	-9.3	-8.7	-10	-10
iterate 2	-18	-18	-18	-18	-12	-9.8	-9.7	-11	-10	-10	-9.6	-10	-10	-9.5	-10	-9.5
iterate 3	-18	-18	-18	-18	-18	-11	-8.8	-9.8	-8.8	-9.6	-10	-10	-11	-11	-11	-10
iterate 4	-18	-18	-18	-18	-18	-18	-11	-10	-9.8	-9.7	-10	-8.9	-8.3	-9.7	-9.7	-10
iterate 5	-18	-18	-18	-18	-18	-18	-18	-11	-8.6	-9.2	-10	-10	-10	-9.1	-9	-9.5
iterate 6	-18	-18	-18	-18	-18	-18	-18	-18	-12	-9.7	-10	-10	-9.8	-8.6	-9	-8.1
iterate 7	-18	-18	-18	-18	-18	-18	-18	-18	-18	-12	-10	-11	-11	-10	-8.7	-10
iterate 8	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-12	-11	-10	-10	-9.5	-10
iterate 9	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-11	-12	-11	-10	-9.1
iterate 10	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-12	-12	-11	-11
iterate 11	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-12	-11	-11
iterate 12	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-12	-12
iterate 13	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-13
iterate 14	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18	-18

Table 2.6: Error of the i-th iterate in the fixed point algorithm for the j-th order of the polynomial. The error is computed taking the mean value of the error of the coefficient in the j-th order.

order	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
iterate 0	-18	-17	-12	-12	-9.8	-11	-11	-9.3	-7.7	-8.1	-8	-9.4	-9.8	-12	-13	-12
iterate 1	-18	-18	-17	-17	-11	-9.6	-9.6	-9.8	-9.5	-9.6	-9.8	-9	-8.6	-8.9	-10	-9.8
iterate 2	-18	-17	-17	-17	-17	-17	-17	-17	-12	-11	-11	-12	-9.7	-11	-11	-11
iterate 3	-18	-18	-18	-17	-17	-17	-17	-17	-17	-17	-17	-17	-17	-17	-17	-17

Table 2.7: Error of the i-th iterate in Newton's method for the j-th order of the polynomial. The error is computed taking the mean value of the error of the coefficient in the j-th order.

2.6.2 Recurrent inversion of polynomials

The two basic identities to be used for the determination of the recurrence formulas for the computation of the inverse polynomial Q of P are:

$$P(Q(\vec{\xi})) = \vec{\xi}, \quad Q(P(\vec{\xi})) = \vec{\xi},$$

which can be rewritten as:

$$\sum_{k\in\mathbb{N}^l} a_k \left(\sum_{k'\in\mathbb{N}^l} b_{k'} \vec{\xi}^{k'} \right)^k = \vec{\xi}, \quad \sum_{k'\in\mathbb{N}^l} b_{k'} \left(\sum_{k\in\mathbb{N}^l} a_k \vec{\xi}^{k} \right)^{k'} = \vec{\xi}. \tag{2.14}$$

Taking into account that the coefficients of P are known and the ones of Q unknown, it seems more useful to use the second formula, since in this way we avoid the computation of powers of unknown variables, that will make the problem more complicated. In this second formula all the unknown coefficients appear only once and not powered to any multi-index k.

To get the recursion formulas we will proceed as in Section 2.2, looking to the coefficients in the above equation order by order.

- Order zero. Looking at the zero order terms of (2.14), the following equation is obtained:

  $$b_0\, a_0 + \sum_{k'} b_{k'}\, a_0^{k'} = 0.$$

  Therefore, it is not possible to determine the value of any $b_{k'}$ since all of them appear in the previous equation. If we impose the condition $a_0 = b_0 = 0$ the problem is solved. As it has already been discussed, this condition is just a translation of the of the functions according to their independent terms.

- Order one. Collecting the terms of order one of the second equation (2.14), we get l^2 different equations. For each component i of the polynomial Q, the following formulae are obtained:

  $$\sum_{|k'|=1} b_{k'}\, a_{k,k'} = k]_i \quad \text{for } |k| = 1,$$

  where $a_{k,k'}$ is the coefficient of order k of the polynomial $P^{k'}(\vec{\xi})$.

  In general, the previous formulae define l systems of equations (one for each component of the polynomial) with l variables $(b_{k'}]_i)$ and l equations (one for each k of order 1).

- Order n. In this case, it is possible to distinguish two different sources of coefficients of order exactly n. On one hand, there are the coefficients coming from the powers of the lower orders already computed. On the other hand, those coefficients that are unknowns that come from a term of the form $b_{k'} P^{k'}(\vec{\xi})$.

As before, the resulting formula after collecting terms of order k in each component i of the polynomial vector formula is:

$$\sum_{|k'|<n} b_{k'} \left(P(\vec{\xi})\right)^{k'} \Bigg]_k + \sum_{|k'|=n} b_{k'}\, a_{k,k'} = 0 \quad \text{for } |k| = n.$$

Again, we obtain l systems of equations (one for each component of the polynomial vector) with $\phi_{n,l}$ variables ($b_{k'}$) and equations (one for each term k).

Solving the preceding systems of equations order by order it is possible to find the inverse polynomial in which we are interested.

Implementation note. Whenever a value b_k is obtained, it is interesting to compute the value of $b_k P(\vec{\xi})^k$ and add it to a residual polynomial $R(\vec{\xi})$. This polynomial will contain the contribution of each term b_k to the higher order terms, i.e. the ones that we want to cancel in subsequent steps. In addition, at the end of each step n the coefficients of the residual polynomial $R(\vec{\xi})$ of order smaller or equal to n will be zero (in machine precision), since they were already cancelled. In further steps, to determine the value of the independent term of the linear equation, since it is already determined in the residual polynomial $R(\vec{\xi})$, we only need to take the coefficient of the corresponding order that we are isolating.

2.6.3 Numerical results

The polynomial that has been used to test the precision of the local inversion is the one obtained using the Jet Transport for the integration of the simple pendulum, $P(\vec{\xi}) \approx \phi(t; t_0, x_0 + \vec{\xi})$. The main advantage of this selection is that it is easy to obtain $Q(\vec{\xi}) = P^{-1}(\vec{\xi}) \approx \phi(t_0; t, \phi(t; t_0, x_0) + \vec{\xi})$. Table 2.8 shows the initial polynomial $P(\vec{\xi}) \approx \phi(2; 0, (0, 4) + \vec{\xi})$ and its inverse computed using the fixed point method.

In order to check the correctness of the inverse polynomial Q, we have done the composition of Q with P and vice-versa. Table 2.9 shows the results of both compositions, $P(Q(\vec{\xi}))$ and $Q(P(\vec{\xi}))$. As expected, in both cases the resulting polynomial is the identity plus some errors. Observe that the errors are of the order of the machine precision for the low order terms. For higher order terms, the error propagation produces an increase of their values. Furthermore, all the implemented methods for the computation of $Q(\vec{\xi})$ give the same result, up to machine precision. Table 2.10 shows the differences between the inverse polynomials computed comparing the results obtained using the fixed point method, Newton's method and the recursive formulae. As before, in all the cases the differences are of the order of the machine precision.

As it is usual in the Jet Transport computations, some estimates of the radius of validity of Q have been done, in other words, an estimation of the maximum value of $\vec{\xi}$ for which the composition $P(Q(\vec{\xi})) = \vec{\xi}$ is under an acceptable tolerance. Observe that, since $Q(\vec{\xi})$ is not an infinite polynomial, the composition will have a remainder, i.e. Q is the inverse of P up to a certain order N, which means that $P(Q(\vec{\xi})) = \vec{\xi} + o(\vec{\xi}^N)$.

The maximum initial box of the polynomial $Q(\vec{\xi})$ is not giving this information, since it only provides the range were $Q(\vec{\xi})$ is computed with good accuracy, but not where its composition with $P(\vec{\xi})$ is good. We do not only want to know about the radius of convergence, but the zone where both polynomials are one the inverse of the other. Figure 2.26 shows the differences $P(Q(\vec{\xi})) - \vec{\xi}$ and $Q(P(\vec{\xi})) - \vec{\xi}$, in log-scale, for different values of $\vec{\xi}$ and for two different jet orders 5 and 15. As it is expected, the size of the region where the jet inverse is valid is higher when the jet has higher order.

As it has been said, if the polynomial $P(\vec{\xi})$ is the expansion of the flow $\phi(t; 0, \vec{x}_0 + \vec{\xi})$, then its inverse can be computed using backwards propagation, this is, propagating backwards the point $\phi(t; 0, x_0)$ up to the initial time, in such way that $Q(\vec{\xi}) = \phi(0; t, \phi(t; 0, \vec{x}_0) + \vec{\xi})$. In that way the resulting polynomial should be the same than the one obtained through inversion algorithms. The inverse polynomials $Q(\vec{\xi})$ computed in this way will be denoted by $Q_{fp}(\vec{\xi})$ and $Q_b(\vec{\xi})$, respectively.

Table 2.11 shows the difference between the inverse computed using the fixed point algorithm and $Q_b(\vec{\xi})$. Although the order one terms coincide, the differences for the higher orders are much larger.

Figure 2.26 also shows, the inversion errors $P(Q_b(\vec{\xi})) - \vec{\xi}$ and $Q_b(P(\vec{\xi})) - \vec{\xi}$ in logarithmic scale for values of $\vec{\xi}$ uniformly distributed in the unit square. There, it can be seen that the range of points where the inverse is accurately computed is wider than the one computed using the inversion algorithms. However, Table 2.11 reflects the result of the composition of both polynomials $Q_b(P(\vec{\xi}))$. There it can be seen that the composition of the polynomials is not giving the identity up to order N in the jet and with machine precision for the lower orders.

A possible explanation for that fact is that in the computation of Q_b, the jet propagation takes into account the dynamics of the problem. However, the inversion algorithm does not detect it, and, furthermore, P has some errors already accumulated. Therefore, when computing the inverse polynomial using the presented algorithms it is not really the inverse of the desired polynomial but an approximation.

Table 2.12 gives the time required for the computation of the inverse polynomial, using jets of order 5, 15 and 20, with each method. The CPU times displayed in the table are the average of 1000 computations with the inversion procedures. For all the orders considered, the fastest method is the one that uses the recursive formulae. This fact is expected, since the number of operations involved in this method is the smallest one.

And interesting fact appears when we compare the CPU time required by the fixed point method and Newton's method; for low orders, the fixed point method is faster than Newton's method. This has sense, since the number of evaluations required by Newton's method is high and in both methods we are doing a few number of iterations. However, when we increase the order of the jets that are used, Newton's method has better time results; again, this fact has sense, since in very few iterations we get the correct inverse polynomial, each Newton iteration will be slower than the one of the fixed point algorithm, but since we are doing much less iterations, the final time is better for Newton's method.

$P_1(\vec{\xi})$	$P_2(\vec{\xi})$	k_1	k_2	$Q_1(\vec{\xi})$	$Q_2(\vec{\xi})$
– ordre 0 –				– ordre 0 –	
0	0	0	0	0	0
– ordre 1 –				– ordre 1 –	
0.95699618	-0.23627904	1	0	0.50603453	0.23627902
2.1827028	0.50603457	0	1	-2.1827026	0.95699609
– ordre 2 –				– ordre 2 –	
0.24330334	0.024168993	2	0	-0.041918209	0.001854564
-0.11274064	-0.11922214	1	1	0.16501876	0.21960203
-0.22847836	-0.049742406	0	2	0.095615685	-0.58500604
– ordre 3 –				– ordre 3 –	
-0.0073809507	0.019505875	3	0	0.095630567	-0.034186404
-0.13284701	0.23290184	2	1	0.01162865	-0.057165278
0.015749553	0.6113075	1	2	-0.14648468	0.059968244
0.088038442	0.45531035	0	3	0.066764758	0.19213266
– ordre 4 –				– ordre 4 –	
-0.029821082	0.026172193	4	0	-0.0018783939	-0.013019214
0.023877776	0.16116774	3	1	-0.041532904	0.052109731
0.142641	0.15007648	2	2	0.028122281	0.10310709
0.1098905	-0.13628115	1	3	0.039942524	-0.30517232
0.027299395	-0.13953506	0	4	-0.04914789	0.18297986
– ordre 5 –				– ordre 5 –	
0.0034427024	0.010773078	5	0	-0.0059390254	0.0037673338
0.040117141	-0.0068272077	4	1	0.014127961	0.0061659244
0.034007495	-0.18316766	3	2	0.0054524673	-0.074618505
-0.039044987	-0.32112265	2	3	-0.038698536	0.05961114
-0.062356042	-0.18450148	1	4	0.034339853	0.046444673
-0.024766882	-0.042778115	0	5	-0.0025472781	-0.085480979
– ordre 6 –				– ordre 6 –	
0.0038009726	-0.0041185678	6	0	0.0012443046	-0.00012198352
0.0019384175	-0.047915622	5	1	0.00068495743	-0.013659354
-0.031549365	-0.10037923	4	2	-0.012282402	0.039111067
-0.056049425	-0.0033759276	3	3	0.017098873	0.0016373722
-0.028780744	0.14592926	2	4	0.0040607923	-0.096717022
0.0048946843	0.13909868	1	5	-0.018665303	0.10283868
0.0056918536	0.04375971	0	6	0.0084265943	-0.021864803
– ordre 7 –				– ordre 7 –	
-0.0002540926	-0.003519269	7	0	-7.8997918e-05	-0.00046455086
-0.0077165247	-0.009794516	6	1	-0.0016680194	0.0043241621
-0.016656112	0.030870905	5	2	0.0053279973	0.0010149694
-0.00065515197	0.1180511	4	3	-0.0023331664	-0.032094178
0.028211519	0.11498872	3	4	-0.010105365	0.051488063
0.028851141	0.010883118	2	5	0.012315561	0.00022280066
0.0097163114	-0.044613662	1	6	-0.0033550739	-0.044708113
0.00073807531	-0.019480104	0	7	-0.0013357457	0.024286997

Table 2.8: Output polynomial resulting of propagating the initial condition (4,0) in a simple pendulum using the Jet Transport, $P(\vec{\xi}) \approx \phi(2; 0, (0, 4) + \vec{\xi})$ (left) and inverse of this polynomial computed using the fixed point method, $Q(\vec{\xi}) = P^{-1}(\vec{\xi}) \approx \phi(t_0; t, \phi(t; t_0, x_0) + \vec{\xi})$ (right). The two central columns of the table give the multi-index terms of each coefficient.

$(P \circ Q)_1(\vec{\xi})$	$(P \circ Q)_2(\vec{\xi})$	k_1	k_2	$(Q \circ P)_1(\vec{\xi})$	$(Q \circ P)_2(\vec{\xi})$
– ordre 0 –			– ordre 0 –		
0	0	0	0	0	0
– ordre 1 –			– ordre 1 –		
1	0	1	0	1	0
4.4408921e-16	1	0	1	0	1
– ordre 2 –			– ordre 2 –		
3.469447e-18	0	2	0	0	6.9388939e-18
1.6653345e-16	1.3877788e-17	1	1	-2.7755576e-17	-6.9388939e-18
0	4.1633363e-17	0	2	-1.110223e-16	4.8572257e-17
– ordre 3 –			– ordre 3 –		
-4.1633363e-17	-3.469447e-18	3	0	6.9388939e-18	-3.469447e-18
-5.5511151e-17	-1.3877788e-17	2	1	-5.5511151e-17	-2.7755576e-17
2.220446e-16	-5.5511151e-17	1	2	0	-2.220446e-16
-8.3266727e-17	5.5511151e-17	0	3	4.4408921e-16	-3.3306691e-16
– ordre 4 –			– ordre 4 –		
-6.9388939e-18	8.6736174e-19	4	0	-1.3877788e-17	0
1.3877788e-17	-6.9388939e-18	3	1	5.5511151e-17	5.5511151e-17
1.6653345e-16	-2.7755576e-17	2	2	0	-1.6653345e-16
4.4408921e-16	-2.220446e-16	1	3	-1.110223e-16	-1.110223e-16
-1.110223e-15	-4.4408921e-16	0	4	1.6653345e-16	-8.3266727e-17
– ordre 5 –			– ordre 5 –		
1.7347235e-18	8.6736174e-19	5	0	-1.0408341e-17	1.7347235e-18
6.9388939e-18	1.0408341e-17	4	1	-4.8572257e-17	3.469447e-18
-8.3266727e-17	-2.7755576e-17	3	2	-2.220446e-16	9.7144515e-17
1.6653345e-16	-2.4980018e-16	2	3	-4.4408921e-16	3.3306691e-16
-7.2164497e-16	-3.3306691e-16	1	4	-3.8857806e-16	4.4408921e-16
1.1310397e-15	-1.110223e-16	0	5	-2.220446e-16	8.3266727e-17
– ordre 6 –			– ordre 6 –		
-1.0842022e-18	-5.4210109e-19	6	0	0	1.7347235e-18
-6.9388939e-18	-1.7347235e-18	5	1	0	0
2.7755576e-17	6.9388939e-18	4	2	-1.110223e-16	0
-2.3592239e-16	-2.7755576e-17	3	3	-3.8857806e-16	-1.3877788e-16
-6.6613381e-16	2.220446e-16	2	4	6.6613381e-16	5.5511151e-17
2.1094237e-15	-3.3306691e-16	1	5	1.5543122e-15	-4.4408921e-16
3.3306691e-16	-2.220446e-16	0	6	9.9920072e-16	-7.2164497e-16
– ordre 7 –			– ordre 7 –		
-6.505213e-19	-1.0842022e-19	7	0	2.6020852e-18	-8.6736174e-19
1.0408341e-17	-4.3368087e-19	6	1	6.9388939e-18	-1.7347235e-17
6.2450045e-17	-6.9388939e-18	5	2	0	-1.110223e-16
1.9428903e-16	-2.7755576e-17	4	3	2.220446e-16	-1.110223e-16
1.6653345e-16	6.2450045e-17	3	4	4.9960036e-16	-6.6613381e-16
1.110223e-15	-5.5511151e-16	2	5	0	-2.220446e-16
1.9428903e-15	2.220446e-16	1	6	-8.8817842e-16	6.1062266e-16
-2.4147351e-15	-4.4408921e-16	0	7	-5.2735594e-16	1.0998147e-15

Table 2.9: Compositions of the polynomial P with its inverse Q $P(Q(\vec{\xi}))$ (left) and $Q(P(\vec{\xi}))$ (right). The two central columns give the multi-index term of each coefficient.

$(Q_{fp} - Q_{rf})_1(\vec{\xi})$	$(Q_{fp} - Q_{rf})_2(\vec{\xi})$	k_1	k_2	$(Q_{fp} - Q_{new})_1(\vec{\xi})$	$(Q_{fp} - Q_{new})_2(\vec{\xi})$
– ordre 0 –				– ordre 0 –	
0	0	0	0	0	0
– ordre 1 –				– ordre 1 –	
0	-2.7755576e-17	1	0	0	0
0	-1.110223e-16	0	1	4.4408921e-16	1.110223e-16
– ordre 2 –				– ordre 2 –	
6.9388939e-18	-4.5536491e-18	2	0	6.9388939e-18	-2.3852448e-18
8.3266727e-17	-5.5511151e-17	1	1	2.7755576e-17	5.5511151e-17
5.5511151e-17	-1.110223e-16	0	2	-8.3266727e-17	1.110223e-16
– ordre 3 –				– ordre 3 –	
1.3877788e-17	1.3877788e-17	3	0	-2.7755576e-17	-6.9388939e-18
-3.469447e-17	4.8572257e-17	2	1	1.0234869e-16	-6.2450045e-17
-2.220446e-16	1.0408341e-16	1	2	1.9428903e-16	-5.5511151e-17
-1.9428903e-16	8.3266727e-17	0	3	-3.6082248e-16	-2.7755576e-17
– ordre 4 –				– ordre 4 –	
-3.0357661e-18	1.7347235e-18	4	0	-2.6020852e-18	0
-2.0816682e-17	6.2450045e-17	3	1	4.1633363e-17	-6.2450045e-17
-2.7755576e-17	-4.1633363e-17	2	2	1.0408341e-17	6.9388939e-17
-6.9388939e-17	-7.7715612e-16	1	3	4.9266147e-16	2.7755576e-16
1.9428903e-16	1.4155344e-15	0	4	1.4571677e-16	-8.0491169e-16
– ordre 5 –				– ordre 5 –	
5.2041704e-18	-2.6020852e-18	5	0	-7.8062556e-18	2.6020852e-18
1.2143064e-17	-1.8214596e-17	4	1	-5.3776428e-17	1.5612511e-17
-4.9439619e-17	5.5511151e-17	3	2	6.0715322e-18	1.3877788e-17
3.3306691e-16	-5.5511151e-17	2	3	3.9551695e-16	-1.8041124e-16
4.1633363e-17	2.9143354e-16	1	4	-1.3877788e-17	-2.3592239e-16
-2.2616457e-15	-1.4988011e-15	0	5	4.1329787e-16	4.7184479e-16
– ordre 6 –				– ordre 6 –	
-2.1684043e-19	-1.4230154e-18	6	0	-8.6736174e-19	1.490778e-18
-8.0230961e-18	1.7347235e-18	5	1	-5.6378513e-18	3.469447e-18
-9.7144515e-17	1.179612e-16	4	2	1.7694179e-16	-1.179612e-16
-4.510281e-17	1.2490009e-16	3	3	2.0816682e-17	-1.8279649e-16
1.0217521e-15	-8.4654506e-16	2	4	-1.6393137e-16	-6.9388939e-17
-3.042705e-15	-6.800116e-16	1	5	2.2655489e-15	-4.3021142e-16
1.6618651e-15	5.6829541e-15	0	6	-2.4112656e-16	5.8286709e-16
– ordre 7 –				– ordre 7 –	
-1.8973538e-19	1.0842022e-18	7	0	-2.1684043e-19	0
2.1684043e-18	-3.469447e-18	6	1	0	1.7347235e-18
-4.5970172e-17	-4.7704896e-18	5	2	-4.9439619e-17	1.6046192e-17
2.992398e-17	-2.220446e-16	4	3	1.0668549e-16	1.3877788e-16
4.510281e-16	-3.469447e-17	3	4	-5.7419347e-16	2.8449465e-16
-3.8684334e-16	-2.0242055e-16	2	5	4.0245585e-16	2.2659825e-16
1.0842022e-17	5.1902926e-15	1	6	8.5131555e-16	1.5126789e-15
4.4430605e-16	-9.5652652e-15	0	7	1.9515639e-18	-1.3912482e-15

Table 2.10: Difference between the inverted polynomial $Q(\vec{\xi})$ computed using the fixed point method Q_{fp} and the recursive formulae Q_{rf} (left); and difference between the inverted polynomial $Q(\vec{\xi})$ computed using the fixed point method Q_{fp} and Newton's method Q_{new}.

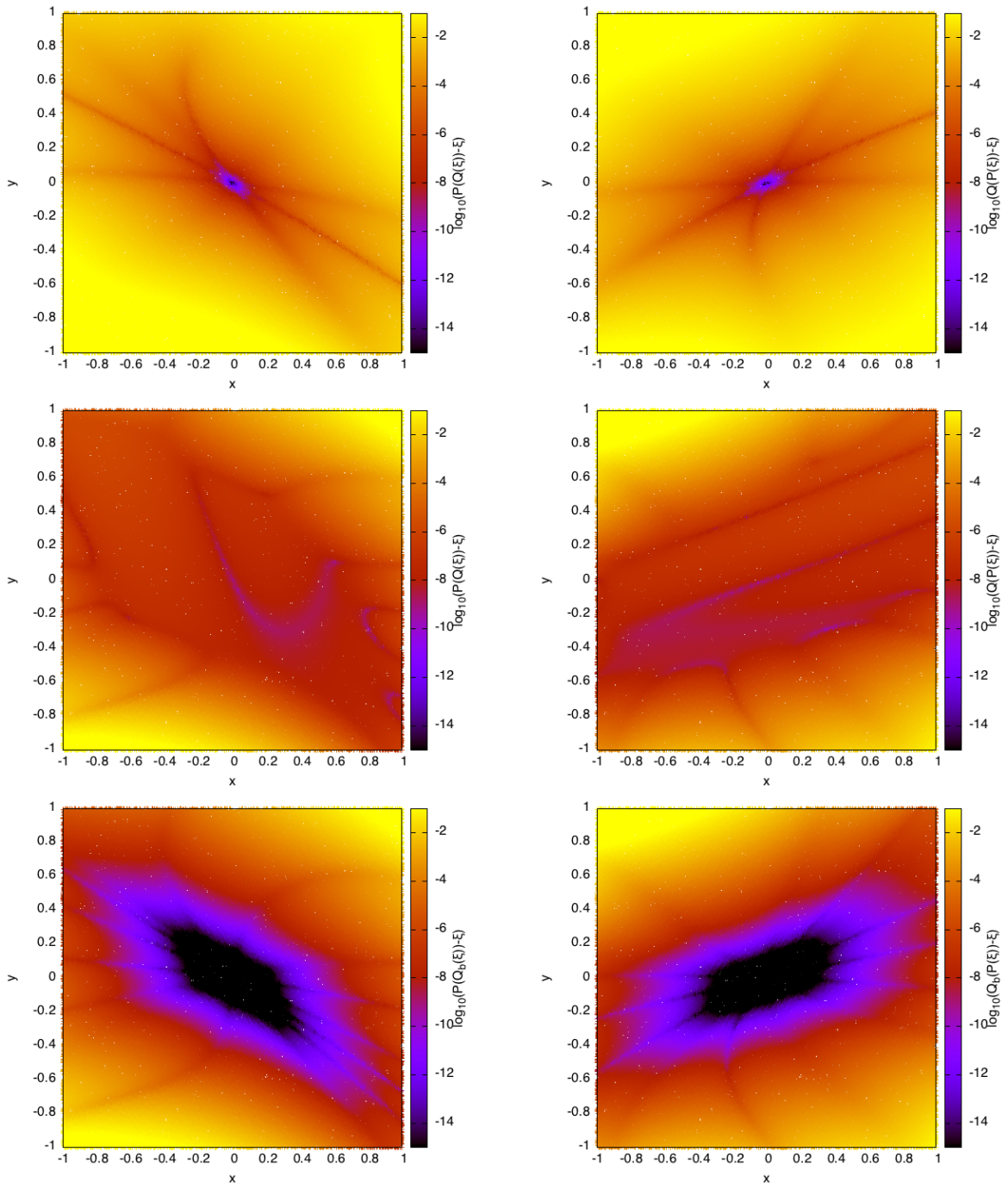


Figure 2.26: Error of the compositions $P(Q(\vec{\xi})) - \vec{\xi}$ and $Q(P(\vec{\xi})) - \vec{\xi}$, taking as initial polynomial $P(\vec{\xi})$ the one obtained using Jet Transport for the integration of a simple pendulum, and as $Q(\vec{\xi})$ the inverse of $P(\vec{\xi})$ computed using the recursive formulae. Top: logarithm of $P(Q_{rf}(\vec{\xi})) - \vec{\xi}$ (left) and $Q_{rf}(P(\vec{\xi})) - \vec{\xi}$ (right) for different values of $\vec{\xi}$ and a maximum order for the jets of 5. Middle: same differences using a maximum order for the jets of 15. Bottom: error of the composition of $P(\vec{\xi})$ with the inverse computed using the Jet Transport propagating backwards in time $Q_b(\vec{\xi})$, $\log_{10} P(Q_b(\vec{\xi})) - \vec{\xi}$.

$(Q_{fp} - Q_b)_1(\vec{\xi})$	$(Q_{fp} - Q_b)_2(\vec{\xi})$	k_1	k_2	$(Q_b \circ P)_1(\vec{\xi})$	$(Q_b \circ P)_2(\vec{\xi})$
– ordre 0 –				– ordre 0 –	
0	0	0	0	0	0
– ordre 1 –				– ordre 1 –	
9.5322631e-08	-8.2485084e-08	1	0	1.0000002	-1.3398779e-07
-2.553495e-07	2.3298672e-07	0	1	7.8845301e-08	0.99999994
– ordre 2 –				– ordre 2 –	
-7.2217722e-08	4.4981002e-08	2	0	-3.9976234e-07	2.9469792e-07
9.889606e-07	-7.3925732e-07	1	1	2.309657e-07	-2.3899005e-07
-2.2752323e-06	1.8052085e-06	0	2	1.5657231e-07	-1.3271042e-07
– ordre 3 –				– ordre 3 –	
-5.1875867e-08	5.3997041e-08	3	0	-2.6481979e-08	1.2104095e-07
3.0272472e-07	-4.4161121e-07	2	1	-1.2428962e-06	1.0180484e-06
1.2832381e-06	-2.5223371e-07	1	2	-6.9338332e-08	-1.4435148e-08
-4.7942516e-06	2.9757492e-06	0	3	1.4335287e-07	-1.2972071e-07
– ordre 4 –				– ordre 4 –	
2.8215459e-08	-1.8672457e-08	4	0	7.7644306e-07	-6.2887397e-07
-1.0789508e-06	8.4054499e-07	3	1	9.1634848e-07	-4.6846545e-07
7.2241725e-06	-6.0690692e-06	2	2	-1.3419575e-06	1.2437764e-06
-1.4436155e-05	1.3183014e-05	1	3	-3.9423224e-07	2.9266859e-07
7.1094127e-06	-7.7871428e-06	0	4	6.1965524e-08	-6.1661477e-08
– ordre 5 –				– ordre 5 –	
-2.7959824e-08	2.1670823e-08	5	0	-6.2664842e-07	3.4368659e-07
3.9982954e-07	-1.9895575e-07	4	1	1.2330621e-06	-1.2546597e-06
-5.2846585e-06	3.0057338e-06	3	2	2.1819745e-06	-1.5829866e-06
2.8945997e-05	-1.898334e-05	2	3	-4.2713729e-07	5.5269488e-07
-6.1227189e-05	4.3353505e-05	1	4	-3.5616186e-07	3.1110826e-07
4.4745701e-05	-3.3357197e-05	0	5	4.0569725e-09	-7.2847927e-09
– ordre 6 –				– ordre 6 –	
3.8193779e-08	-2.7337708e-08	6	0	-1.7207459e-07	2.369635e-07
-3.6139568e-07	2.4092393e-07	5	1	-2.1194199e-06	1.478077e-06
2.6870207e-07	4.9611819e-07	4	2	-1.6135159e-07	-4.0023985e-07
1.1653522e-08	-5.1824481e-06	3	3	1.9484874e-06	-1.6911709e-06
1.928462e-05	1.5813157e-06	2	4	3.3469832e-07	-1.7570005e-07
-6.2968973e-05	2.4651556e-05	1	5	-1.2578113e-07	1.281449e-07
5.5860875e-05	-2.9583024e-05	0	6	-6.5751251e-09	5.3611604e-09
– ordre 7 –				– ordre 7 –	
-5.2571252e-09	-9.3725151e-10	7	0	4.9317356e-07	-3.1870229e-07
1.3316384e-07	-3.650492e-08	6	1	6.8693948e-07	-1.4240636e-07
1.8116901e-07	-3.1925946e-07	5	2	-2.1572527e-06	1.8736004e-06
-1.3271695e-05	1.017246e-05	4	3	-1.5485352e-06	8.9289912e-07
7.1895558e-05	-5.6392388e-05	3	4	6.8195682e-07	-7.776045e-07
-0.00015383518	0.00012729325	2	5	3.8811587e-07	-3.306141e-07
0.00014162481	-0.00012703631	1	6	1.4876918e-08	-5.6149872e-09
-4.5025806e-05	4.666104e-05	0	7	-1.580216e-10	2.1094898e-10

Table 2.11: Left columns: Differences between the inverse polynomial computed using the fixed point algorithm $Q_{fp}(\vec{\xi})$ and the backward integration using the Jet Transport $Q_b(\vec{\xi})$. Central columns: values of the multi-indices for the corresponding coefficients. Right columns: Value of the composition between the backwards integration polynomial $Q_b(\vec{\xi})$ and the forward integration polynomial $P(\vec{\xi})$.

Method	Fixed Point	Newton	Recursive formulae
order 5	$0.396 \cdot 10^{-3}$	$0.918 \cdot 10^{-3}$	$0.068 \cdot 10^{-3}$
order 15	0.393	0.344	$4.388 \cdot 10^{-3}$
order 20	2.18	1.805	0.019145

Table 2.12: CPU time (in seconds) for the different polynomial inversion methods: the fixed point algorithm, Newton's method and the recursive formulae, for orders 5, 15 and 20. The CPU times are the average values of 1000 inversions.

2.7 Mapping density functions

When doing experiments and observations it is usual to have some uncertainty in the results obtained. These uncertainties can be due to different sources, for instance: errors in the calibration of the tools used, human errors, or errors due to the procedures used to get the final data. In order to take into account these errors, the actual state of an experiment is given in two terms: the first one is the observed quantity, the second one is the error that this quantity has. A classical method to describe the errors, of the resulting data of a certain system, is by means of a probability distribution function, such as a Gaussian one, in such way that one does not know which is the actual state of the system but the probability of being in a certain set of states.

Assume that the role of a certain system is just to perform a change of variables, $\vec{y} = f(\vec{x})$, according to a given law f, in this situation it is possible to know which are the changes produced by f if the values of the samples $\vec{x}$ have no errors (deterministic samples), however, it is not straight forward to know what happens with the errors of the output values $\vec{y}$ of the system.

A relevant situation corresponds the case in which $\vec{x}$ is an event of a certain random variable X, with Gaussian distribution of mean μ and variance matrix Σ, and the change of variables is a linear function $f(\vec{x}) = A + B\vec{x}$. In this situation it is well known that the final distribution is again a Gaussian distribution with mean $A + B\mu$ and covariance matrix $B\Sigma B^*$.

In the case of systems whose changes are driven by an ordinary differential equation, it is usual to consider the first variational equations of the system:

$$\begin{cases} \dot{\vec{x}} = f(\vec{x}), \\ \vec{x}(0) = \vec{x}_0, \\ \dfrac{d}{dt} D_{\vec{x}_0}\vec{x} = D_{\vec{x}} f(\vec{x}) D_{\vec{x}_0}\vec{x}, \\ D_{\vec{x}_0}\vec{x}(0) = Id, \end{cases}$$

and taking a distribution centred at $\vec{x}_0$, we have a linear map from the initial to the final state

$$f(\vec{\xi}) = \phi(t; t_0, \vec{x}_0) + D_{\vec{x}_0}\phi(t; t_0, \vec{x}_0)\vec{\xi}.$$

This introduces a linear change of variables with $A = \phi(t; t_0, \vec{x}_0)$ and $B = D_{x_0}\phi(t; t_0, \vec{x}_0)$. Therefore, a random variable $\vec{\xi} \sim \mathcal{N}(\mu = 0, \Sigma)$ will be propagated to a random variable $\vec{\eta} = f(\vec{\xi})$ described by a Gaussian distribution with average $\phi(t; t_0, \vec{x}_0)$ and covariance matrix $\Sigma_Y = D_{\vec{x}_0}(t; t_0, \vec{x}_0)\sigma D_{\vec{x}_0}(t; t_0, \vec{x}_0)^*$.

Doing that, it is possible to obtain a first order approximation of the final density function. However, if the non-linear effects of the problem are important, the previous approach may not work. The objective of this section is to show how to incorporate the higher order terms in the density propagation using the Jet Transport. In what follows, the algorithms to compute the density up to high orders will be presented, as well as some test to check the results.

2.7.1 Propagation of high order densities

The probability density function f (PDF) of a random variable X gives the probability for an event $\vec{x}$ of the variable X to be in a certain region $\mathcal{A}$ computing the integral of f in $\mathcal{A}$

$$P(\vec{x} \in \mathcal{A}) = \int_{\mathcal{A}} f(\vec{x}) dx.$$

After a change of variables $\vec{y} = g(\vec{x})$, the probability must be preserved, i.e. the probability that the random variable X takes a value in $\mathcal{A}$ must be the same that the random variable Y takes a value in $\mathcal{B} = g(\mathcal{A})$, $P(\vec{x} \in \mathcal{A}) = P(g(\vec{x}) = \vec{y} \in g(\mathcal{A}))$, this is:

$$P(\vec{y} \in g(\mathcal{A})) = P(\vec{x} \in \mathcal{A}) = \int_{\mathcal{A}} f(\vec{x}) dx = \int_{g(\mathcal{A})} f(g^{-1}(\vec{y}))|\det D_y g^{-1}(\vec{y})| dy.$$

Therefore the PDF, $\tilde{f}(\vec{y})$, of the random variable Y will be:

$$\tilde{f}(\vec{y}) = |\det D_y g^{-1}(\vec{y})|\, f(g^{-1}(\vec{y})). \tag{2.15}$$

In order to apply the previous formula the function g must be invertible (it must define a change of variables), otherwise, when the change is not global, the last integral must be decoupled into the different regions where each determination of g^{-1} has sense. Further discussion on this topic will come later.

In the situations that will be considered, the change of variables g is given by the flow map $\phi(t; t_0, \vec{x}_0)$. With the usual methods for the numerical integration of differential equations it is not possible to know the function ϕ since they only provide the value of ϕ at $\vec{x}_0$. Using the Jet Transport it is possible to get the Taylor expansion of ϕ around $\vec{x}_0$ up to the desired order, which gives a local approximation of ϕ.

Assume that $\phi(t; t_0, \vec{x}_0 + \vec{\xi}) \approx P(\vec{\xi}) = \sum_{|k|\leq n} a_k \vec{\xi}^k$. If the PDF at time t_0 is $f(\vec{\xi})$, then the approximate PDF at the final time t_f will be $\tilde{f}(\eta) \approx \left|\det D_\eta P^{-1}(\eta)\right| f(P^{-1}(\eta))$.

This approximation suggests the following algorithm for the computation of $\tilde{f}$:

1. Compute the approximated polynomial flow map $P(\vec{\xi}) \approx \phi(t; t_0, \vec{x}_0 + \vec{\xi})$.
2. Compute the inverse of $P(\vec{\xi})$.
3. Compute the Jacobian of $P^{-1}(\vec{\eta})$.
4. Complete the composition $f(P^{-1}(\vec{\eta}))$ and multiply it by the Jacobian.

In previous sections we have already seen how to compute the inverse of a Taylor expansion, here it is also possible to compute the inverse of P by computing the Taylor expansion of $\phi(t_0; t, \phi(t_0; t, \vec{x}_0) + \vec{\xi})$.

Since we are working with polynomials, the Jacobian of the change of variables is easy to compute differentiating each polynomial with respect to the corresponding variable. The computation of the determinant of the differential matrix is done using a LU polynomial decomposition.

Coming back to the injectivity of the change of variables, in the problem under consideration this is not a big issue because all the operations done are local. The polynomial P may not be injective, in fact, in most of the cases they will be not, however, we are not interested in a global inversion of P but in a local one, close to some point $\vec{x}_0$. The point $\vec{x}_0$ is the one that has most of the density around it, therefore, most of the density will be propagated by the local inversion. In other words, the pre-image of $\mathcal{B}$ can contain values far away from $\vec{x}_0$, however, the probability given by the density function f in those pre-images will be small since, in general, the density function considered will be centred at $\vec{x}_0$ that is the point given by the observation.

2.7.2 Testing the results

In order to test if the resulting PDF has been well determined, we can compute a numerical approximation of $\tilde{f}$ at $\vec{y}$ according to the following algorithm:

1. Consider 2^n points around $\vec{y}$, $\vec{y}_k = \vec{y} + \sum(-1)^{k_i} \dfrac{\vec{e}_i}{2}\delta$, that will be the corners of a n-dimensional box around $\vec{y}$, where $k = (k_1, \ldots, k_n)$ is a multi-index whose coefficients are equal to 1 or 0, and $\vec{e}_i$ is the unitary Euclidean vector in the i-th direction.
2. Compute the pre-images of the points $\vec{y}_k$ by the function g. These points define an n-dimensional box $\mathcal{R}$.
3. Compute the probability under the density function f: $p_{\mathcal{R}} = P(\vec{x} \in \mathcal{R}) = \int_{\mathcal{R}} f(\vec{x}) d\vec{x}$.
4. The probability of being in $\mathcal{B}$ is then $p_{\mathcal{B}} = \dfrac{p_{\mathcal{R}}}{\delta^n}$.

The idea behind this numerical procedure is to see which is the probability of being in a small set $\mathcal{B}$ around the desired point $\vec{y}$ computing the probability of being in a set $\mathcal{R}$, in the original coordinates, where we know the probability density function, and assuming that the all the points in $\mathcal{B}$ has the same density (or almost the same one).

Remark:

- When we compute the pre-images of the points $\vec{y}_k$, we need to take all the possibles k. At a first glance, it seems a good option to take $\vec{y}_{(1,\dots,1)}$ and $\vec{y}_{(0,\dots,0)}$ to determine the region $\mathcal{R}$, i.e. to take the two points separated by a diagonal. However, the region determined by those pre-images may be far away from the real pre-image region and, therefore, introduce big errors. Hence, a more general procedure should be established.

- In order to compute the integral in the region $\mathcal{R}$, a change of variables must be done to get good variables. A straightforward option is the following: given the points $g^{-1}(\vec{y}_k) = (x_{k,1}, \dots, x_{k,n})$ use consecutive parametrisations as follows:

  First, introduce the parametric coordinate λ_1 to move along the first direction

  $$s_{(1,k^1)}(\lambda_1) = \lambda_1 x_{1,k^1} + (1-\lambda_1)x_{0,k^1}, \quad k^1 \in \mathbb{Z}_2^{n-1},$$

  where k^1 are the multi-indices with $n-1$ coefficients 0 or 1. In the next step consider

  $$s_{(2,k^2)}(\lambda_1,\lambda_2) = \lambda_2 s_{(1,(1,k^2))}(\lambda_1) - (1-\lambda_2)s_{(1,(0,k^2))}(\lambda_1) \quad k^2 \in \mathbb{Z}_2^{n-2}.$$

  Again, k^2 are multi-indices with $n-2$ coefficients 0 or 1. The s_2 family is parametrising the set of 2-dimensional sides of the box $\mathcal{R}$. In general the i-th step defines the parametrisations:

  $$s_{i,k^i}(\lambda_1,\dots,\lambda_i) = \lambda_i s_{(i-1,(1,k^i))}(\lambda_1,\dots,\lambda_{i-1}) + (1-\lambda_i)s_{(i-1,(0,k^i))}(\lambda_1,\dots,\lambda_{i-1}) \quad k^i \in \mathbb{Z}_2^{n-i}.$$

  Which parametrises the i-th dimensional face.

  The final parametrisation is given by $s(\lambda_1,\dots,\lambda_n) = s_{n,\emptyset}$.

- Since we are doing a small deviation, it is reasonable to take linear parametrisations of the domain of integration.

2.7.3 Examples

Next we will show some examples to illustrate the procedure. In the first one, that illustrates the one-dimensional case, a Normal distribution $\mathcal{N}(0,1)$ in one dimension is propagated with the function $y = x^2 + 3x + 2$. The 2-dimensional situation is illustrated with the propagation of densities in the simple pendulum model $\ddot{x} = -\sin(x)$. A last example using Kepler's problem is introduced to work with the density of points in a more general situation.

The one-dimensional case

Assume that X is a random variable with Gaussian distribution, $X \approx \mathcal{N}(0,1)$. After some computations, we apply the change of variables $y = g(x) = x^2 + 3x + 2$. In this case it is easy to compute the inverse of g: $g_{\pm}^{-1}(y) = \dfrac{-3 \pm \sqrt{1+4y}}{2}$ so, in this case there are two different inverses. Since the random variable X is centred at zero, the g_+^{-1} solution is the one in which the PDF will have more area concentrated. Therefore, using (2.15) we get

$$\tilde{f}_+(y) = \frac{\exp\left(-(-3+\sqrt{1+4y})^2/8\right)}{\sqrt{2\pi}} \cdot \frac{2}{\sqrt{1+4y}}.$$

Observe that using the previous density we are only taking into account the right branch of the parabola described by $g_+^{-1}(x)$. The left branch is computed using $g_-^{-1}(x)$

$$\tilde{f}_-(y) = \frac{\exp\left(-(-3-\sqrt{1+4y})^2/8\right)}{\sqrt{2\pi}} \cdot \frac{2}{\sqrt{1+4y}}.$$

Therefore the total density will be:

$$\tilde{f}(y) = \tilde{f}_+(y) + \tilde{f}_-(y) = \frac{2\left(\exp\left(\frac{-(-3+\sqrt{1+4y})^2}{8}\right) + \exp\left(\frac{-(-3-\sqrt{1+4y})^2}{8}\right)\right)}{\sqrt{2\pi}\sqrt{1+4y}}.$$

All these operations can also be done using the polynomial algebra. First we introduce the function g as a jet. In that case, since it is a polynomial it will be perfectly represented, as long as we use jets of order at least 2. Then, we use one of the inversion algorithms introduced in section 2.6 to compute g^{-1}. Observe that the inversion algorithm will only give one branch. In general, it will not be possible to obtain all the branches, because the approximation obtained is around only one of them. Next, we compute the normal distribution using the jets to get a polynomial approximation of it around the point of interest. Finally, we do the composition of the two functions and we multiply by the Jacobian of g^{-1}. At the end we obtain a polynomial P_d that determines the PDF around the inverse point, that in the example is around $y = 2$, i.e. $P_d(\vec{\xi}) = \tilde{f}(2+\xi)$.

Figure 2.27 shows the final density function computed using the analytical expression of $\tilde{f}_+(x)$, its numerical approximation using $\delta = 10^{-3}$ and $\delta = 10^{-4}$ and evaluating the polynomial $P_d(\xi)$ using jets of orders 20 and 60. The figure also shows the linear propagation of the density.

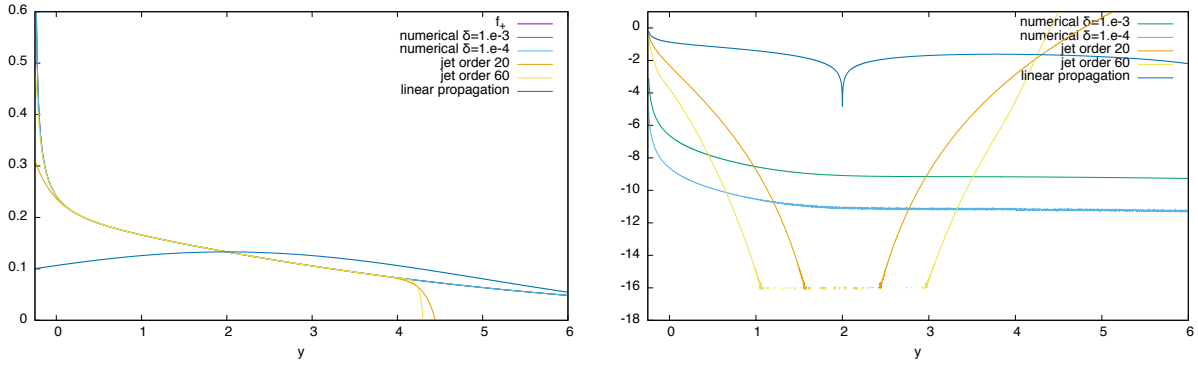


Figure 2.27: In the left plot, final density function computed using the analytical expression of $\tilde{f}_+(x)$ (purple), its numerical approximation using $\delta = 10^{-3}$ (green) and $\delta = 10^{-4}$ (blue) and evaluating the polynomial $P_d(\vec{\xi})$ using jets of orders 20 (orange) and 60 (yellow), together with the linear propagation of the density. The right hand side displays, using the same colors, the $\log_{10}$ of the difference between the 5 last methods and the analytical expression of the final density. .

From the above figure, one can see that the numerical approximation is good up to some error due to the precision of the approximation, taking smaller values of δ better approximations are obtained. The approximation obtained using polynomial algebra is also giving good results, in fact are better than the resulting from the numerical approach around the point $y = 2$. Higher orders of the jet give better approximations, as it can be expected. The linear propagation of the density gives the right density just at the propagation point and a very small region around it, since the nonlinear effects are important.

The left branch of the parabola is not propagated using these method. The inversion procedures are not able to compute it correctly because the polynomial that we are using to get the approximation is centred at

the other branch; when we try to get the inversion in the left one the procedures does not converge. Figure 2.29 shows the amount of density that we are loosing by not taking into account this left branch (left), as well as the final density propagating both branches and only the right one. We can see that if we are far from the point -0.25, that is the point were the singularity of the change of variables is, the left branch is not affecting the final density and it is only a small portion of it where there is some effect. Figure 2.28 shows the relative weight of the left branch with respect to the total density. We can observe that the weight of the left branch around $y = 2$ is approximately a 0.01. Therefore, the approximation obtained of the density function is good.

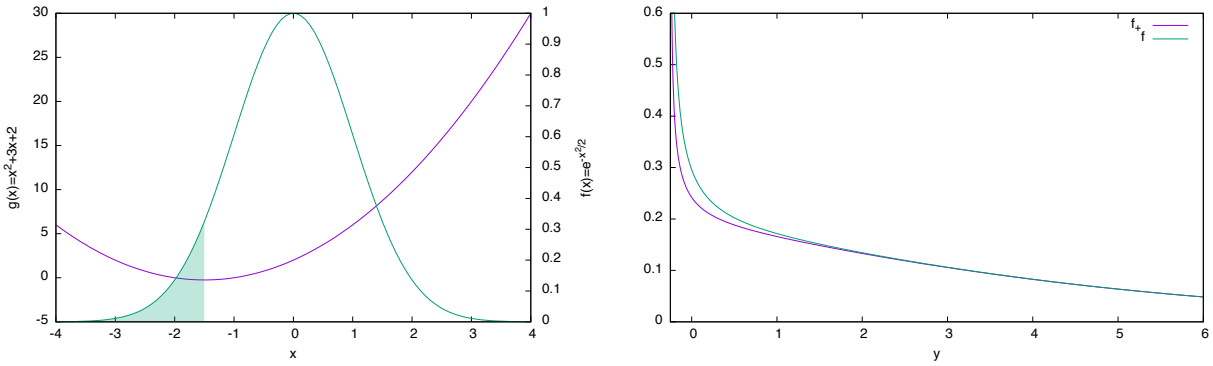


Figure 2.28: Left: initial density function and graph of the change of variables introduced. The shadowed region shows the probability that we are loosing if we do not propagate the left branch of the change of variables. Right: propagated density functions for the polynomial function $g(x) = x^2 + 3x + 2$ using the exact propagation $\tilde{f}$ (green) and the propagation through the same function g of the right branch of the parabola (purple), which contain the propagation of the point with more density in the original coordinates.

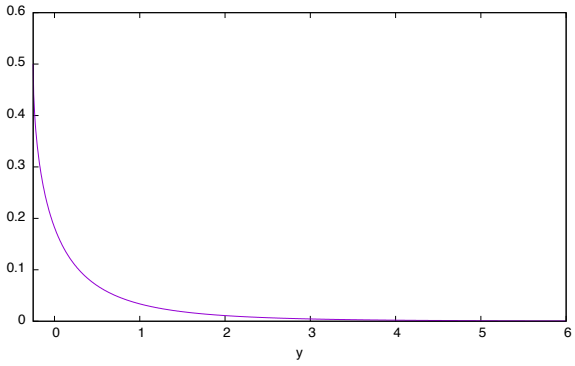


Figure 2.29: Relation between the density coming from the left branch $\tilde{f}_-$ and the total density $\tilde{f}$: $\tilde{f}_-/\tilde{f}$. We can see that contribution of density by the left branch g_- is almost null for values far away from the singular point

The simple pendulum

As a second test example, consider a simple pendulum, an initial condition $\vec{x}_0$ with an error distribution following a normal $\mathcal{N}(\mu, \Sigma)$ with $\mu = (0, 0)$ and $\Sigma = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. The goal is to compute the probability density

function at some time t. We can predict that the average of the new PDF $\tilde{f}$ will be around $y = \phi(t; t_0, x_0)$, but the question is to determine the distribution around it.

Figure 2.30 shows the PDF computed using the Jet Transport methodology. As we can see, the central part reminds a Gaussian function with some deviation, while at some edges it starts to increase in a strange way. This is because, far from the center, the polynomial P is not approximating with enough accuracy the function $\phi(t; t_0, x_0)$. If the probabilities that we need to compute are close to the central point of the propagation the results of the procedure are satisfactory.

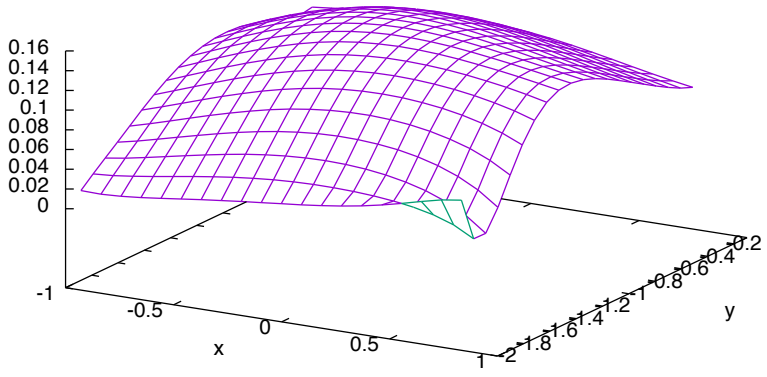


Figure 2.30: Probability density function propagated through the flow map from the starting initial point $x_0 = (1.08, 0.26)$ for a final time $t = 2$. The initial PDF is a $\mathcal{N}(0, Id)$.

Figure 2.31 shows a comparison between the probability density function computed using the Jet Transport and the one computed numerically. We can observe that the central part of the propagation is giving coherent results with the accuracy computed. Again, large discrepancies appear at the edges of the figure, due to the bad approximation that the flow map polynomial is giving there.

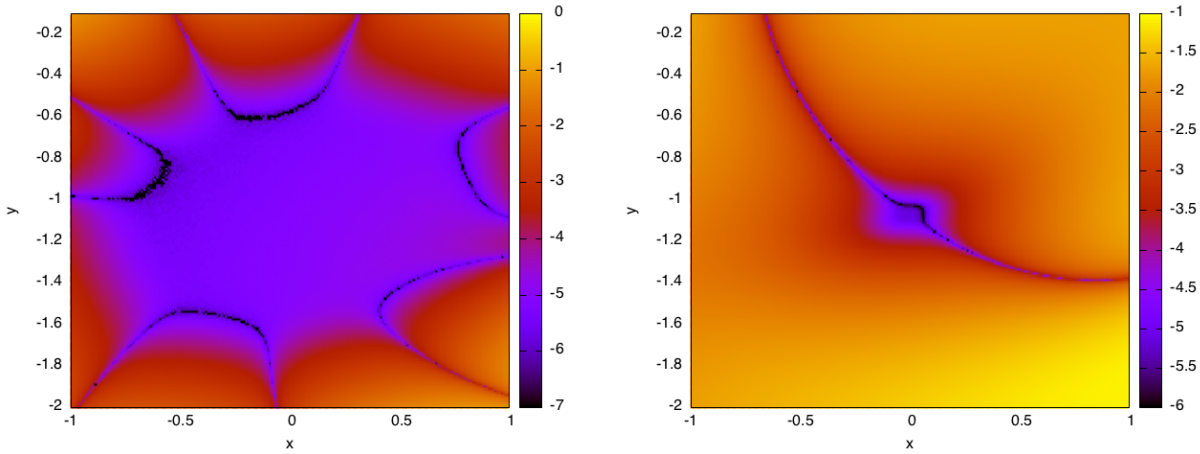


Figure 2.31: Left: Differences in $\log_{10}$ scale between the PDF computed using the Jet Transport and the numerically computed PDF. An initial Gaussian density $\mathcal{N}(0, Id)$ is propagated through the flow map from the starting initial point $x_0 = (1.08, 0.26)$ up to a final time $t = 2$. Right: differences, in a ($\log_{10}$ scale, between the PDF computed using the numerical method and the Gaussian density obtained propagating using the linear flow map.

Figure 2.31 also shows the difference between the linear propagation of the previous density and the numerical results. There, we can see how it approximates the density in a neighbourhood of the central

point.

Chapter 3

Dynamics indicators and LCS

3.1 Introduction

Lagrangian Coherent Structures (LCS), introduced by G. Haller et al. [HY00] for the study of dynamical systems, give a methodology to identify the boundaries between regions in the configuration space with orbits that have different dynamical behaviour.

The first definition of the Lagrangian Coherent Structures was given in [HY00] by means of Finite Time Lyapunov Exponents (FTLE). The underlying idea is to look at the FTLE scalar-field. At each point, $\vec{x}$, of the phase space, the associated FTLE is an indicator of the behaviour of the orbits of the neighbouring points to $\vec{x}$ over a time interval $[t_0, t_0 + T]$ that measures the average separation between trajectories. A low FTLE value at $\vec{x}$ reveals that the orbits of two neighbouring points of $\vec{x}$ ($\vec{x}_1(0)$ and $\vec{x}_2(0)$) will be close each other during that time-interval, while a high FTLE value indicates that the orbits of the neighbouring points will have different behaviours, usually because the orbits will diverge from each other. Figure 3.1 shows an intuitive idea of what kind of orbits we can expect for high and low FTLE.

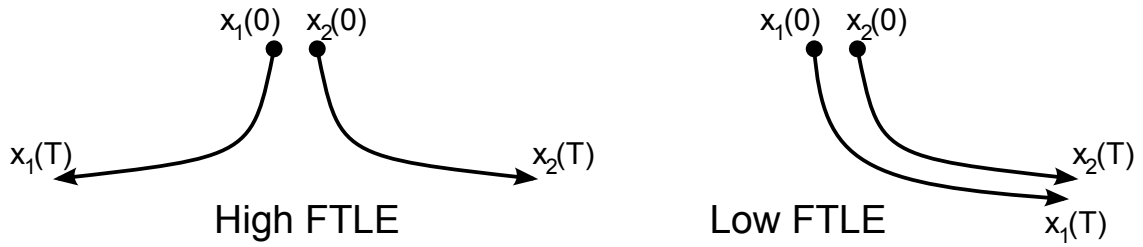


Figure 3.1: Qualitative representation of two trajectories with nearby initial conditions and different FTLE values.

The FTLE are computed using the flow map. Given a dynamical system represented by the ordinary differential equation,

$$\dot{\vec{x}}(t) = f(\vec{x}(t), t),$$

consider the flow map, denoted by $\phi(t; t_0, \vec{x}_0)$ (or $\phi_{t_0}^t(\vec{x}_0)$, whenever the initial and final time are fixed), as the function such that, for an initial condition $\vec{x}_0 \in \mathcal{D} \subset \mathbb{R}^n$ at time t_0, gives the state at time t associated to $\vec{x}_0$:

$$\begin{array}{rcl} \phi : \mathbb{R} \times \mathbb{R} \times \mathcal{D} & \longrightarrow & \mathcal{D} \\ (t, t_0, \vec{x}_0) & \longmapsto & \phi(t; t_0, \vec{x}_0). \end{array}$$

The value of the *Finite Time Lyapunov Exponent* (FTLE) at x_0 from $t = t_0$ up to $t = t_0 + T$ is defined as:

$$\sigma_{t_0}^T(\vec{x}_0) := \frac{1}{|T|} \ln \left\| \frac{\partial \phi_{t_0}^{t_0+T}(\vec{x}_0)}{\partial \vec{x}} \right\|_2, \tag{3.1}$$

this is, the FTLE is the normalised square of the largest eigenvalue of

$$C_{t_0}^{t_0+T}(\vec{x}_0) = \left(\frac{\partial \phi_{t_0}^{t_0+T}(\vec{x}_0)}{\partial \vec{x}} \right)^* \cdot \left(\frac{\partial \phi_{t_0}^{t_0+T}(\vec{x}_0)}{\partial x} \right) \equiv \left(D\phi_{t_0}^{t_0+T}(\vec{x}_0) \right)^* \cdot \left(D\phi_{t_0}^{t_0+T}(\vec{x}_0) \right) \equiv J^* J,$$

where the star indicates transposition. A linear analysis shows that the eigenvector associated to the largest eigenvalue of $C_{t_0}^{t_0+T}(\vec{x}_0)$, which is usually called the *Cauchy-Green tensor*, gives the maximum expansion direction in a neighbourhood of $\vec{x}_0$.

Differences in the propagation behaviour of two close initial conditions can be due to a separating invariant manifold. Hence, the FTLE can be used as indicators of this kind of structures. Unfortunately, hyperbolic invariant manifolds are not the only dynamical structures separating regions. Furthermore, there are some manifolds that are not detected by the LCS method (see [Hal11]). Therefore, using only the FTLE scalar-field it is possible to find false positives and false negatives hyperbolic structures, as it is shown by Haller et al. in [Hal11].

Figure 3.2 shows the FTLE scalar-field for the simple pendulum, the perturbed pendulum, the CR3BP and the ER3BP. There, one can see how the invariant manifolds appear as the ridges (yellow zones) of the FTLE field. But there are also other ridges that do not correspond to any invariant manifold.

For a two-dimensional dynamical system ($n = 2$), a *Lagrangian Coherent Structure* (LCS) with initial and final times t_0 and T, respectively, is an injective curve $\vec{c} = \vec{c}(s) \subset \mathcal{D} \subset \mathbb{R}^2$ parametrised by $s \in (a, b) \subset \mathbb{R}$ such that verifies the following two conditions:

1. $\vec{c}\,'(s)$ is parallel to the gradient of the FTLE field at $\vec{c}(s)$, i.e:

$$\vec{c}\,'(s) \parallel \nabla \sigma_{t_0}^T(\vec{c}(s)) \quad \forall s \in (a, b).$$

2. The unit normal vector $\vec{n}(s)$ to $\vec{c}(s)$ verifies:

$$\vec{n}^T \Sigma \vec{n} = \min_{\|u\|=1} \vec{u}^T \Sigma \vec{u} < 0,$$

   for all $s \in (a, b)$, where Σ is the Hessian of the FTLE field defined by:

$$\Sigma = \Sigma(\vec{x}, t_0, T) = \frac{d^2 \sigma_{t_0}^T(\vec{x})}{d\vec{x}^2}.$$

According to Lekien et al. [LSM07], in higher dimensions, the injective curve $\vec{c}$ is substituted by a codimension 1 orientable differentiable manifold $\mathcal{M} \subset \mathcal{D} \subset \mathbb{R}^n$ such that each point $\vec{x} \in \mathcal{M}$ verifies:

- The Cauchy-Green strain tensor $C_{t_0}^{t_0+T}(\vec{x})$ has $n-1$ eigenvalues less than 1 and one eigenvalue greater than 1.
- The unit normal vector $\vec{n}$ to the manifold $\mathcal{M}$ at $\vec{x}$ is orthogonal to the gradient $\nabla \sigma_{t_0}^T(\vec{x})$.
- If Σ denotes the matrix of second derivatives of $\sigma_{t_0}^T(\vec{x})$, used as a bilinear form evaluated at the point $\vec{x}$, then the two following conditions hold:
  - $\Sigma(\vec{n}, \vec{n}) < 0$,
  - $\Sigma(\vec{n}, \vec{n}) < \Sigma(\vec{u}, \vec{u})$ for all unit vector $\vec{u}$ such that $|\langle \vec{u}, \vec{n} \rangle| \neq 1$.

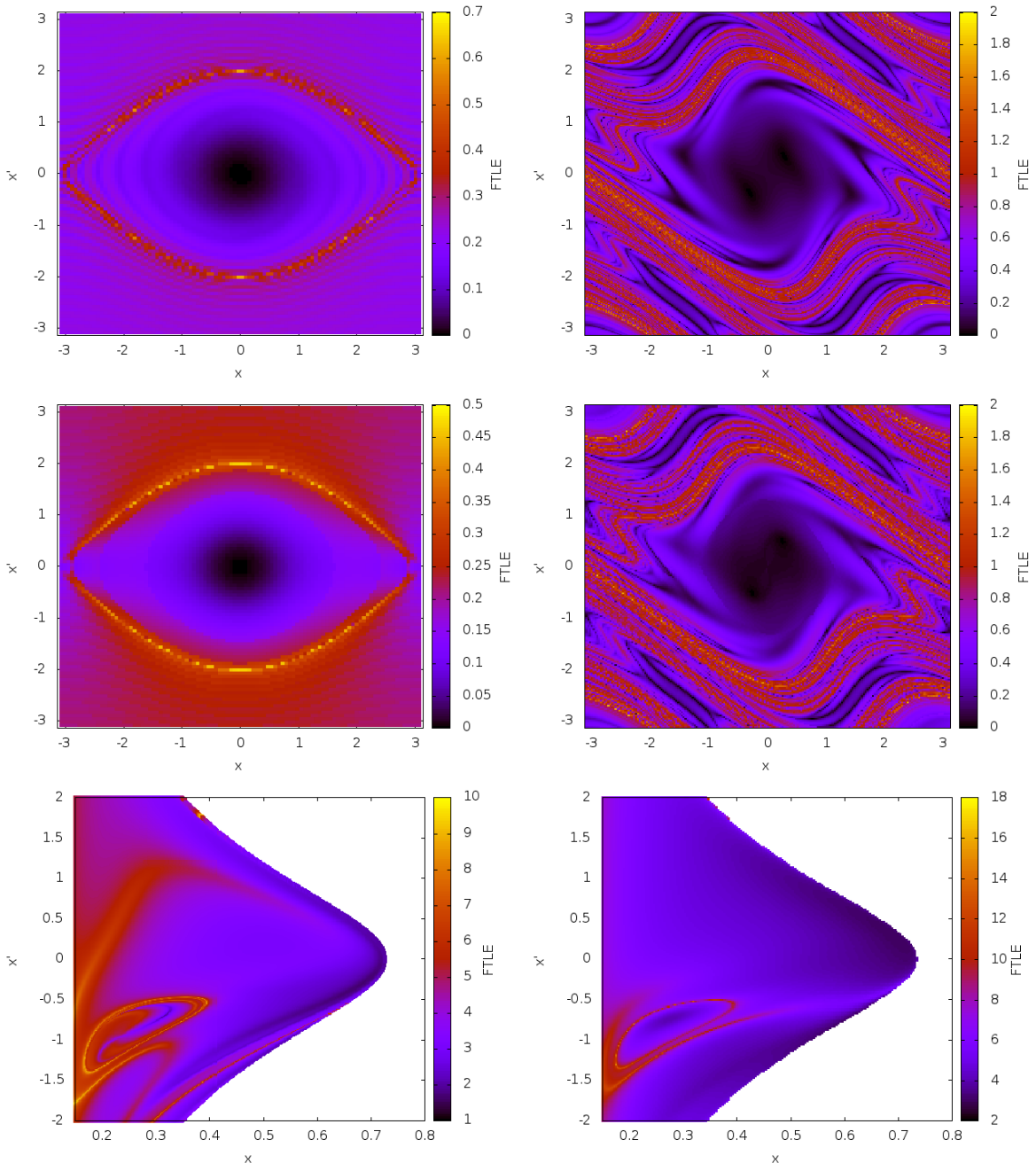


Figure 3.2: Finite time Lyapunov exponents scalar fields for different systems. From left to right and top to bottom (four top figures): simple pendulum and perturbed pendulum with fixed integration time $T = 30$ and $T = 10$ respectively; simple pendulum and perturbed pendulum with a close final condition time around $T = 30$ and $T = 10$ respectively (see section 3.2.1 for further details about the close final condition time). In these four cases the points for the computation of the FTLE are taken in a grid of the $(x_i, \dot{x}_j)$ plane. The two bottom figures correspond to the circular and elliptic restricted 3-body problem for a fixed final time $T = 2$ adimensional units. The mass parameter is $\mu = 0.1$, and in the elliptic case $e = 0.04$, $f_0 = 0$. The value of the energy used to reduce the dimension of the problem is $E_0 = \tilde{E}_0 = -1.725$, and the intersecting condition is given by the hyperplane $y = 0$. The grid of points used for the representation is: $(x_i, 0, \dot{x}_j, \dot{y}(x_i, 0, \dot{x}_j, E_0))$. See section 3.2.1 for further details about the representation of the LCS's.

Observe that the second condition substitutes the first one for two-dimensional LCS. In higher dimensions, the manifold that we are looking for must be tangent to the gradient of the FTLE. The third condition is also equivalent to the second condition of the two-dimenisonal LCS. In both cases it warranties that the manifold follows the ridge of maximum FTLE. The first condition is added in order to avoid degenerate LCS.

Some examples show that not all the LCS are related to invariant manifolds. To deal with these situations, G. Haller et al. [Hal11] introduced the so called *Hyperbolic Lagrangian Coherent Structure* (HLCS) adding additional requirements to the LCS that use the so called normal repulsion rate and repulsion ratio. The HLCS are defined as manifolds that maximise the expansions produced by the flow in the normal direction with respect to the expansions in the tangent direction of the manifold.

A *material surface* $\mathcal{M}(t)$ is defined as the slice, at time t, of the manifold $\mathcal{M}$ generated by the advection of an $(n-1)$-dimensional surface of initial conditions, $\mathcal{M}_0$, by the flow map $\phi(t;t_0,\vec{x}_0)$, i.e. $\mathcal{M}(t) = \phi(t;t_0,\mathcal{M}(t_0))$.

To measure how strongly repelling the material surface $\mathcal{M}(t_0)$ is, at each point $\vec{x}_0 \in \mathcal{M}(t_0)$ a unit normal vector $\vec{n}_0$ to $\mathcal{M}(t_0)$ ($\vec{n}_0 \in N_{\vec{x}_0}\mathcal{M}(t_0)$) is selected, and its evolution is followed under the linearised flow given by $D\phi_{t_0}^{t_0+t}(\vec{x}_0)$. Let $\rho_{t_0}^{t_0+t}(\vec{x}_0,\vec{n}_0)$ be the length of the normal component to $\mathcal{M}(t)$ of $D\phi_{t_0}^{t_0+t}(\vec{x}_0)\vec{n}_0$:

$$\rho_{t_0}^{t_0+t}(\vec{x}_0,\vec{n}_0) = \left\langle \vec{n}_t, D\phi_{t_0}^{t_0+t}(\vec{x}_0)\vec{n}_0 \right\rangle,$$

where $\vec{n}_t$ is the unit normal vector to $\mathcal{M}(t)$ at $\vec{x}_t = \phi_{t_0}^{t_0+t}(\vec{x}_0)$. $\rho_{t_0}^{t}(\vec{x}_0,\vec{n}_0)$ is called the *normal repulsion rate* of $\mathcal{M}(t)$ along the orbit $\phi(t;t_0,\vec{x}_0)$. If is larger (smaller) than 1, then $\mathcal{M}(t)$ has been overall repelling (attracting) between t_0 and t_0+t along the trajectory starting at $\vec{x}_0$.

In order to deal with any possible tangential growth within $\mathcal{M}(t)$ larger than the growth normal to $\mathcal{M}(t)$, the so called *repulsion ratio* is introduced as:

$$\nu_{t_0}^{t}(\vec{x}_0,\vec{n}_0) = \min_{e_0 \in T_{x_0}\mathcal{M}(t_0)} \frac{\left\langle \vec{n}_t, D\phi_{t_0}^{t_0+t}(\vec{x}_0)\vec{n}_0 \right\rangle}{\|D\phi_{t_0}^{t_0+t}(\vec{x}_0)\vec{e}_0\|},$$

where $T_{x_0}\mathcal{M}(t_0)$ denotes the set of unit tangent vectors to $\mathcal{M}(t_0)$ at $\vec{x}_0$. Note that the numerator of the above formula is the projection of the propagation of the normal vector at time t_0 to the normal direction at time t. The denominator is the maximum length of the tangential propagation; since it is transported by the flow, it will still be tangential. Therefore, the repulsion ratio ν gives the relation between the normal and the tangential expansions.

With the above definitions, the material surface $\mathcal{M}(t)$ is said to be *normally hyperbolic over* $[t_0,t_0+T]$ if there exists constants $a,b>0$ such that for all $\vec{x}_0 \in \mathcal{M}(t_0)$ and $\vec{n}_0 \in N_{\vec{x}_0}\mathcal{M}(t_0)$ the following conditions hold:

- $\rho_{t_0}^{t_0+T}(\vec{x}_0,\vec{n}_0) \geq e^{aT}$,
- $\nu_{t_0}^{t_0+T}(\vec{x}_0,\vec{n}_0) \geq e^{bT}$.

A *hyperbolic LCS* is a normally hyperbolic material surface $\mathcal{M}(t)$ such that its normal repulsion rate admits a point-wise non-degenerate maximum value along $\mathcal{M}(t)$ for all locally C^1-close material surfaces.

In a naive way, the HLCS can be seen as manifolds such that the expansion in the orthogonal direction to them is higher than the ones in any of the tangent directions.

Haller et al. [Hal11] give also a characterisation of the hyperbolic LCS by means of the eigenvalues $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$, and their associated eigenvectors $\vec{\xi}_1,\vec{\xi}_2,...,\vec{\xi}_n$, of the Cauchy-Green positive definite matrix:

$$\left(D\phi_{t_0}^{t_0+T}(\vec{x}_0)\right)^* \left(D\phi_{t_0}^{t_0+T}(\vec{x}_0)\right).$$

The result is summarised in the following:

Theorem 3 ([Hal11]). *A compact material surface $\mathcal{M}(t) \subset \mathcal{D}$ over the interval $[t_0,t_0+T]$ is a hyperbolic LCS if and only if the following conditions are verified for all $\vec{x}_0 \in \mathcal{M}(t_0)$:*

1. $\lambda_{n-1}(\vec{x}_0, t_0, T) \neq \lambda_n(\vec{x}_0, t_0, T) > 1$,
2. $\vec{\xi}_n(\vec{x}_0, t_0, T) \perp T_{x_0}\mathcal{M}(t_0)$,
3. $\left\langle \nabla\lambda_n(\vec{x}_0, t_0, T), \vec{\xi}_n(\vec{x}_0, t_0, T) \right\rangle = 0$,
4. *The matrix* $L(\vec{x}_0, t_0, T)$ *defined by*

$$L(\vec{x}_0, t_0, T) = \begin{pmatrix} \nabla^2 C^{-1}[\vec{\xi}_n, \vec{\xi}_n, \vec{\xi}_n, \vec{\xi}_n] & 2\frac{\lambda_n-\lambda_1}{\lambda_1\lambda_n}\left\langle \vec{\xi}_1, \nabla\vec{\xi}_n\vec{\xi}_n \right\rangle & \dots & 2\frac{\lambda_n-\lambda_{n-1}}{\lambda_{n-1}\lambda_n}\left\langle \vec{\xi}_{n-1}, \nabla\vec{\xi}_n\vec{\xi}_n \right\rangle \\ 2\frac{\lambda_n-\lambda_1}{\lambda_1\lambda_n}\left\langle \vec{\xi}_1, \nabla\vec{\xi}_n\vec{\xi}_n \right\rangle & 2\frac{\lambda_n-\lambda_1}{\lambda_1\lambda_n} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 2\frac{\lambda_n-\lambda_{n-1}}{\lambda_{n-1}\lambda_n}\left\langle \vec{\xi}_{n-1}, \nabla\vec{\xi}_n\vec{\xi}_n \right\rangle & 0 & \dots & 2\frac{\lambda_n-\lambda_{n-1}}{\lambda_{n-1}\lambda_n} \end{pmatrix},$$

where C *denotes the Cauchy-Green tensor at* $\vec{x}_0$*, is positive definite for all* $\vec{x}_0 \in \mathcal{M}(t_0)$.

This theorem gives also the following algorithm to detect hyperbolic LCS (see [Hal11] for details).

Algorithm 3.

Step 1. Compute the two larger eigenvalue λ_n and λ_{n-1}, as well as $\vec{\xi}_n$, for all $\vec{x}_0 \in \mathcal{D}$.

Step 2. Determine the solution set Z defined by:

$$Z = \left\{ \vec{x}_0 \in \mathcal{D} \mid \left\langle \nabla\lambda_n(\vec{x}_0, t_0, T), \vec{\xi}_n(\vec{x}_0, t_0, T) \right\rangle = 0 \right\}.$$

Step 3. Identify repelling weak LCS (WLCS) at $t = t_0$ as the subsets of $Z_{WLCS} \subset Z$ such that:

1. $\lambda_{n-1} \neq \lambda_n > 1$,
2. $\vec{\xi}_n \perp T_{x_0} Z_{WLCS}$.

Step 4. Identify repelling LCS as the $(n-1)$-dimensional surfaces $Z_{LCS} \subset Z_{WLCS}$ such that the matrix L is positive definite.

Step 5. Repeat steps 1-4 backwards in time (from $t_0 + T$ to t_0) to obtain attracting LCS.

Step 6. Verify the robustness of the LCS found by checking that $\left\langle \vec{\xi}_n, (\vec{\xi}_n, \nabla^2\lambda_n)\vec{\xi}_n \right\rangle + \left\langle \nabla\lambda_n, (\nabla\vec{\xi}_n)\vec{\xi}_n \right\rangle$ is bounded away from zero.

When the dynamical system has a first integral $H(\vec{x}, t)$, one can consider the material surface $\mathcal{M}$ defined by $\mathcal{M}_c(t) = \{(\vec{x}, t) | H(\vec{x}, t) = c\}$, where c is a certain constant. In this situation, the unitary vector normal to the material surface $\mathcal{M}$ can be described by means of the gradient of the first integral, $\vec{n}_{t_0} = \nabla H(\vec{x}_0, t_0)/\|\nabla H(\vec{x}_0, t_0)\|$. In this way, the value of ρ can be computed as:

$$\rho_{t_0}^{t_0+t}(\vec{x}_0) = \frac{\|\nabla H(\vec{x}_0, t_0)\|}{\sqrt{\left\langle \nabla H(\vec{x}_0, t_0), \left(C_{t_0}^{t_0+t}(\vec{x}_0)\right)^{-1} \nabla H(\vec{x}_0, t_0) \right\rangle}},$$

and the third condition of the above theorem can be replaced by:

$$\left\langle \nabla\rho_{t_0}^{t_0+T}(\vec{x}_0), \nabla H(\vec{x}_0, t_0) \right\rangle = 0. \tag{3.2}$$

3.2 Computation of LCS

This section is devoted to explain the implementation that has been done of the algorithm for the determination of the hyperbolic LCS, a refinement of the procedure, and the results obtained for the different dynamical systems that have been considered.

The first step of the algorithm requires the determination of the eigenvalues λ_i and eigenvectors $\vec{\xi}_i$ of the Cauchy-Green tensor $C = J^*J$; therefore, the first thing that must be computed is the matrix $J = D\phi_{t_0}^{t_0+T}(\vec{x})$. This computation can be done in a number of different ways, for instance:

- Using numerical differentiation for the computation of the derivatives that appear in the components of J. In this way:

$$J = \begin{pmatrix} \left(\frac{\phi_{t_0}^{t_0+T}(\vec{x}+\Delta x_1)-\phi_{t_0}^{t_0+T}(\vec{x}-\Delta x_1)}{2\delta x_1}\right)_{[1]} & \cdots & \left(\frac{\phi_{t_0}^{t_0+T}(\vec{x}+\Delta x_n)-\phi_{t_0}^{t_0+T}(\vec{x}-\Delta x_n)}{2\delta x_n}\right)_{[1]} \\ \vdots & \ddots & \vdots \\ \left(\frac{\phi_{t_0}^{t_0+T}(\vec{x}+\Delta x_1)-\phi_{t_0}^{t_0+T}(\vec{x}-\Delta x_1)}{2\delta x_1}\right)_{[n]} & \cdots & \left(\frac{\phi_{t_0}^{t_0+T}(\vec{x}+\Delta x_n)-\phi_{t_0}^{t_0+T}(\vec{x}-\Delta x_n)}{2\delta x_n}\right)_{[n]} \end{pmatrix},$$

  where the subindex $[i], i = 1, \ldots, n$ denotes the i-th component of the associated vector, and $\Delta x_i = (0, \ldots, \delta x_i, \ldots, 0)$. The computation of the adverted grid, this is: the the values $\phi_{t_0}^{t_0+T}(\vec{x})$ for all the points $\vec{x}$ of the grid, allows the determination of the matrix J without extra integrations, except for some points in the border of the region explored. Note that the adverted grid has all the values $\phi_{t_0}^{t_0+T}(\vec{x} \pm \Delta x_i)$ required for the determination of J. However, the precision of the derivatives will be small and, furthermore, the CPU time required for its computation will be large, except if Δx_j is also large. Of course, the precision can be improved using Richardson interpolation without too much computational effort.

- Integrating the differential equations of the dynamical model together with the associated variational equations. In this way, a linear system of $n \times n$ differential equations, for the components of the matrix J, is added to the original one. In exchange to this increment of equations, usually more precision is obtained than using numerical differentiation. This has been the option selected for the computations shown in this section.

- A third option is to use the Jet Transport, with jets of at least order 1, to propagate the flow map. In this way, the components of the matrix J are determined without adding additional equations; however, the CPU time required is greater than the one need by the previous strategies.

An important issue that must be considered is that when the integration time interval increases, the components of J also increase and, therefore, the accurate computation of its eigenvalues and eigenvectors can be difficult. To overcome this drawback, it is convenient to split the total time interval $[t_0, t_0 + T]$ in N sub-intervals with endpoints at $t_0, t_1, \ldots, t_{N-1}, t_N = t_0 + T$, as well as the power method for the computation of eigenvalues and eigenvectors. In this way the matrix J can be written as the composition of N matrices

$$J = J_{t_0 \to t_N} = J_{t_{N-1} \to t_N} \cdot J_{t_{N-1} \to t_{N-2}} \cdot \ldots \cdot J_{t_1 \to t_2} \cdot J_{t_0 \to t_1},$$

where $J_{t_i \to t_{i+1}} = D\phi_{t_i}^{t_{i+1}}(\vec{x})$. Then, applying the power method step by step the precision of eigenvalues and eigenvectors of C is increased.

The gradient of the greatest eigenvalues field, $\nabla\lambda_n(\vec{x}_0, t_0.T)$, must be computed at all the points of the grid (step 3 of the algorithm). Since the eigenvalues are computed at all the points, the values of $\nabla\lambda_n(\vec{x}_0, t_0.T)$ can be easily obtained using numerical differentiation using the values of $\lambda_n(\vec{x}_0, t_0.T)$ at the grid points, again except at some border points.

3.2.1 Representation of the results in LCS computations

The usual representation of LCS is done displaying the computed FTLE scalar field, this is: a two-dimensional coloured plot such that each point in the plot corresponds to a state (or its 2-dimensional projection) of the phase space. The colour represents the magnitude of the FTLE. In this section it will be shown how to deal with higher dimensions, together with some alternatives to get better representations.

For one, or one and a half, degrees of freedom dynamical systems, like the pendulum or the perturbed pendulum, the full phase space can be represented in a 2-dimensional plot. However, for systems with more degrees of freedom there is no such straightforward option. For two degrees of freedom systems, there are some possibilities to reduce the dimension. A usual one is to use a certain surface of section and, in some cases, a first integral of the differential equations if they exist. In the CR3BP the hyper-plane defined by $y = 0$ together with Jacobi's first integral allows a representation of the phase space (at each energy level) using the $(x, \dot{x})$ plane. In the ER3BP, one can use the pseudo-energy $\hat{E}$ to reduce the dimension. Of course, one can use different hyper-planes as surfaces of section and try to extract from them the desired structure.

The top left plot of Figure 3.2 shows the FTLE field for the pendulum. One can see that, in addition to the main yellow curve associated to the invariant manifolds of the unstable equilibrium point, there appear some extra (light) curves over and under the invariant manifolds. To explain the appearance of these curves, recall that all the motions of the pendulum are periodic (mod 2π). Then, when the value of T for the computation of the FTLE is fixed, there are points in the phase space that at $t = T$ are again at its initial state (i.e. the points with a period dividing the final time T) while for other initial conditions the final state is far from the initial one. These last ones are responsible of the false structures. To avoid them to appear, the final time T must be selected as a function of the initial condition in such way that the initial and final states coincide, or, at least, are as close as possible. The second row of Figure 3.2 has been obtained using this modified final time. Now, all the "false positives" obtained using a fixed final time have disappeared. In return, the resulting plots are no longer continuous, showing discontinuities when the number of revolutions done by the pendulum changes.

The above strategy is not feasible for the perturbed pendulum. In this case not all the orbits are periodic and, therefore, there is no a final time T for which the initial and final states coincide. Instead, one can look for a time $\tilde{T}$ such that in the time interval $[0.8\,T, 1.2\,T]$ the initial and final states are close enough. For the simple pendulum this procedure gives the same result obtained when a suitable value of T for each initial condition is used, since the distance between the final and initial states is zero, but for the perturbed pendulum there are no major differences when they are compared with the results obtained using a fixed value of T.

As in the simple pendulum case, the high FTLE ridges obtained for the periodically perturbed pendulum correspond to invariant manifolds. When the perturbation is added, the hyperbolic fixed point of the simple pendulum becomes a periodic orbit with the period of the perturbation. This periodic orbit has stable and unstable invariant manifolds that are the ones detected by the high values of the FTLE. Furthermore, at the top and bottom part of the image there are high FTLE values corresponding to a part of the invariant manifolds of other two periodic orbits with period twice the one of the perturbation. Finally, there are also low FTLE regions (dark regions) corresponding to stable periodic orbits. Figure 3.3 shows two examples of such kind of periodic orbits. The first one is located in one of the top most black zone in the central stable region, the other one is in the bottom right of the stable region. Both initial conditions can be identified in the top plot of Figure 3.3.

Invariant manifolds and LCS associated to periodic orbits around L_1

First of all, it must be noted that, since the phase space of the CRTBP has dimension four, in order to visualise the results for this dynamical model some of the reductions mentioned in the preceding in section 3.2.1 must be done. Some of them have already been used by Gawlik et al. [GMDTC09] for the elliptic CRTBP. There, they introduce a mixed position-velocity space using x and $\dot{x}$ as the two dimension space, fix $y = 0$ and determine $\dot{y}$ according to the value of the selected Jacobi constant. Another feasible strategy is to use the Periapsis Poincaré Mapping and compute the LCS for that particular map (see instance Short et al. [SH14]).

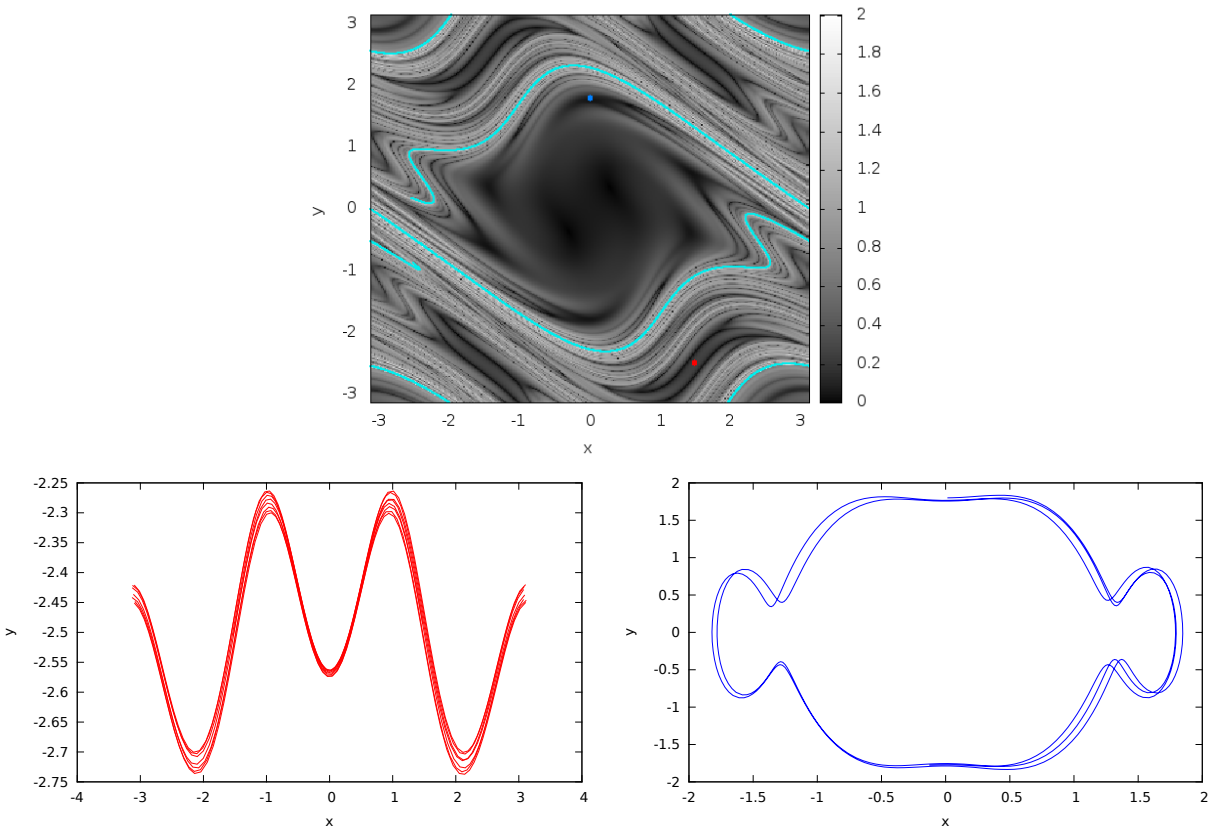


Figure 3.3: Two examples of orbits in stable regions close to periodic orbits with period twice (left) and six times (right) the period of the perturbation. The top figure shows the location of the initial conditions of both orbits (near (1.5,-2.5) and (0.0,1.5), red and blue points, respectively). The cyan curves in the top plot shows the invariant manifold of the periodic points and the invariant manifolds of the periodic orbit with period the one of the perturbation in the upper and bottom part.

In the computations that follow two different surfaces of section have been used: the first one is the hyperplane $x = x_0$ (for some x_0 fixed and usually small), in this case $(y, \dot{y})$ are the two free variables of the system. For a fixed value of the Jacobi constant, C_0, one can determine $\dot{x}$ from:

$$C(x_0, y, \dot{x}, \dot{y}) = C_0. \tag{3.3}$$

An advantage of using this hyperplane as surface of section is that the hyperbolic invariant manifolds associated to the periodic orbits that will be considered are transversal to it. Hence, one can determine their intersections with this hyperplane without problems (tangencies) as is shown in Fig. 3.4.

The second section that has been used is the hyperplane $y = 0$, now the free variables are $(x, \dot{x})$ and $\dot{y}$ is computed in such a way that the Jacobi's constant remains constant for all the points in the grid used for the computation of the FTLE.

Removing some spurious regions

Figure 3.4 shows, in the $(y, \dot{y})$ plane, the regions with high values of the FTLE field, as well as the first two intersections, with $x = 0$ (top) and $x = 0.2$ (bottom), of the invariant unstable manifold of the periodic orbit associated to L_1 for $C_0 = 3.1638 = C(L_1) - 0.035$. There is a good agreement between the intersections of

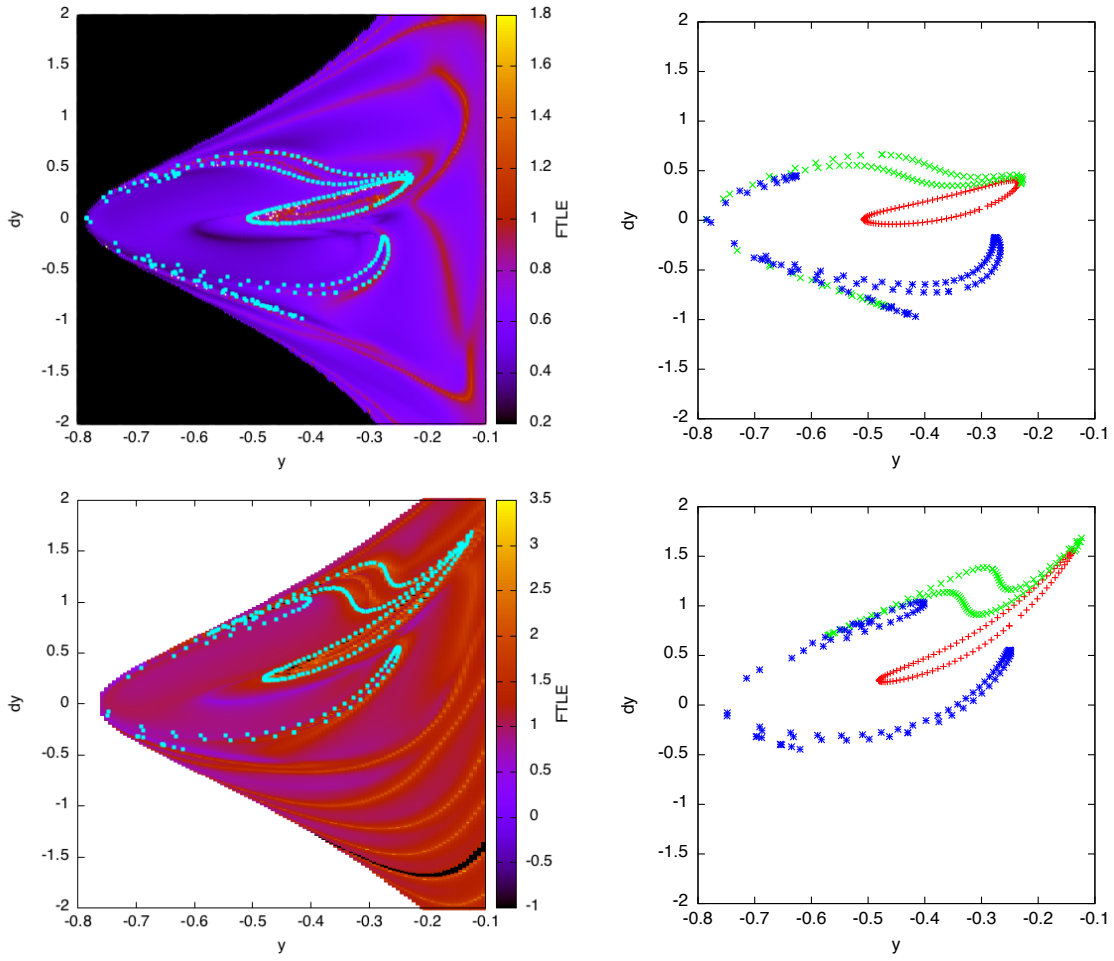


Figure 3.4: For $C(x, y, \dot{x}, \dot{y}) = 3.1638$ and $x = 0$ (top) and $x = 0.2$ (bottom), $(y, \dot{y})$ FTLE field together with the first two intersections of the unstable invariant manifold of the Lyapunov periodic orbit about L_1 for this value of the Jacobi constant (left hand side figures). In the right hand side figures, only the intersections of the manifolds with the two hyperplanes have been represented.

the manifolds with some of the regions with high FTLE values. The computation of the FTLE has been done using a final time $T = -10$ on a regular grid with 200×200 points.

As it is clear from the figure, some other regions with high FTLE values appear that can not be easily identified with intersections of other unstable manifolds. We will refer to them as *false positives*. The number and distribution of the false positives changes when the intersecting hyperplane $x = x_0$ does. This is also clear in the same figure, in which the FTLE field is displayed using the hyperplanes $x_0 = 0$ and $x_0 = 0.2$, in both cases using the same value of the Jacobi constant, C_0, the final time T of integration, and the step-sizes in the grid.

This false positives are due to the procedure used for the computation of the FTLE. Recall that the FTLE give an indication of the variations (sensitivity) of the final conditions of orbits at $t = T$ with respect to their initial conditions. Large values for this sensitivity may be due to the fact that an hyperbolic manifold is acting as a separatrix of two close initial conditions or to some other effect. In the case of these false positives detected in Figure 3.4 they are due to close approaches of the massless particle to the large primary.

In Figure 3.5 we have represented both the distance of the massless particle to the large primary at $t = T$ (lower curve) together with the FTLE values (for the same value of T) of orbits with initial conditions along

the line $(x, y, \dot{x}, \dot{y}) = (0.2, y, \dot{x}_c, -7y-3)$, where the value of $\dot{x}_c$ has been chosen such that the Jacobi constant of all them is equal to $C_0 = 3.1638$. From this figure it clearly follows that the false positives are associated to close approaches to the large primary. To remove these false positives, one option is to compute the FTLE at the apoapsis $t = T_a$. To do this, we first integrate the equations of motion, together with their variational equations, from time t_0 up to time T. Once this time is reached we continue the integration until the next apoapsis of the orbit. Observe that now the final time of the computation is not constant for all the points of the grid, as it happens with the pendulum.

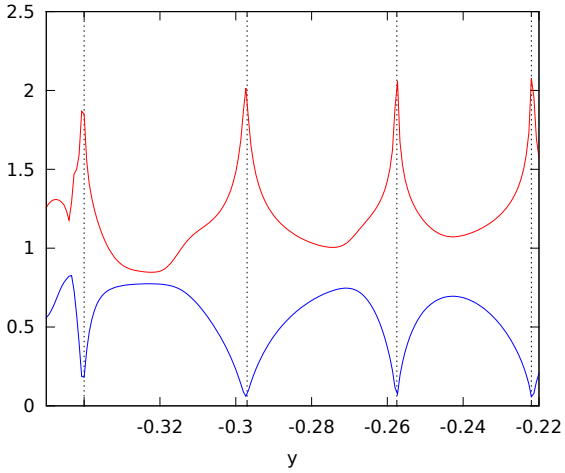


Figure 3.5: FTLE (upper curve) and final distance to the big primary (lower curve) for initial conditions of the form $(x, y, \dot{x}, \dot{y}) = (0.2, y, \dot{x}_c, -7y - 3)$

Therefore two similar orbits can look different at time T due to the fact that one of them is close to one of the primaries while the other is still far. The first orbit will be, after some time, far from the primary but not at time T, so its seems convenient to allow variations of the final time.

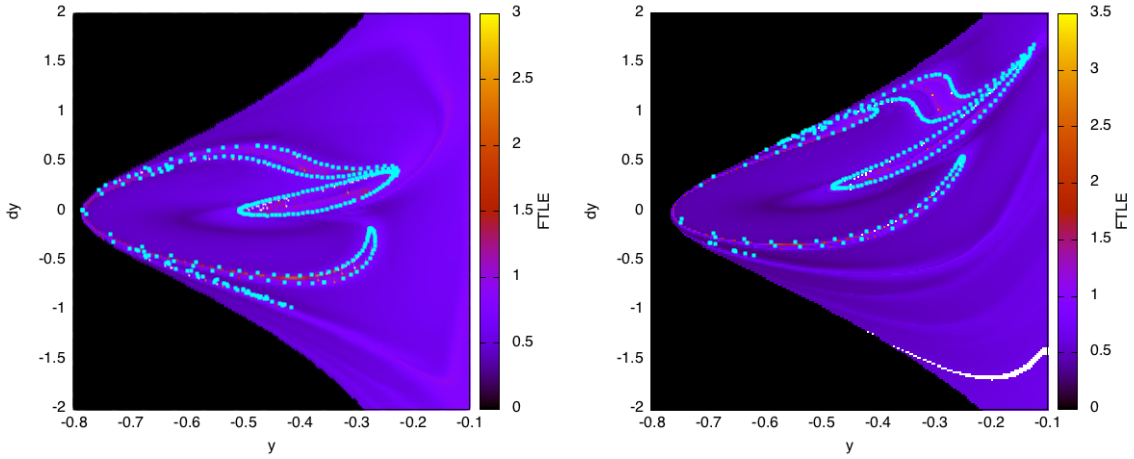


Figure 3.6: (y-$\dot{y}$) FTLE field, computed using the apoapsis strategy, with $x = 0.0$ (left) and $x = 0.2$ (right) for $C(x, y, \dot{x}, \dot{y}) = 3.1638$, together with the first two intersections of the unstable invariant manifold with the hyperplane $x = x_0$.

Figure 3.6 shows the FTLE field using the apoapsis strategy, there one can see that the false positives

appearing in Figure 3.4 have almost disappeared.

3.2.2 Iterative computation of the hyperbolic LCS

Once the false positives have been removed, one can proceed with the hyperbolic LCS computation algorithm 3. Besides the determination of the highest eigenvalues, the first thing to do is to compute the set Z of Step 2 of the algorithm, defined by:

$$Z = \left\{ \vec{x}_0 \in \mathcal{D} \mid \left\langle \nabla \lambda_n(\vec{x}_0, t_0, T), \vec{\xi}_n(\vec{x}_0, t_0, T) \right\rangle = 0 \right\}.$$

Note that this set is defined by means of an equality so, in the numerical computations, it is substituted by:

$$Z_\delta = \left\{ \vec{x}_0 \in \mathcal{D} \mid \left| \left\langle \nabla \lambda_n(\vec{x}_0, t_0, T), \vec{\xi}_n(\vec{x}_0, t_0, T) \right\rangle \right| < \delta \right\},$$

where δ is a certain tolerance.

The value of δ must be selected a priory, although in an iterative implementation of the algorithm scheme it can be modified at each iteration. In this case, the strategy that has been used is to take a relatively large initial value for δ (i.e. $\delta_1 = 0.1$) and then, at each iteration, decrease it by a factor r: $\delta_{i+1} = r \cdot \delta_i$. In what follows, the values $r = 0.8$ or $r = 0.9$ are the ones that have been most frequently used. In this way, the first iterations of the algorithm are less restrictive in the selection of the points of Z_δ, when the diameter of the grid is large; later, it will be explained how the diameter of the mesh is determined. In this way, some of the points detected in Z_δ during the first iterations are refined in the subsequent ones using finer meshes, and the others discarded.

Figure 3.7 (left) shows, superimposed to the FTLE field, the set $Z_{0.1}$ for a grid of $200\times$ 200 points inside the square of the $(y\text{-}\dot{y})$ plane of size $[-0.8, 0.0] \times [-2.0, 2.0]$. When the the condition of Step 3 ($\lambda_n > 1$) is added, the set of points in $Z_{0.1}$ is reduced to the ones displayed in the right-hand side plot of the same figure. The points that fulfil both conditions remain on the intersections of the manifold of the Lyapunov periodic orbit, nevertheless there are still very few points verifying both conditions. Of course, if the diameter of the mesh is reduced then the set of points in $Z_{0.1}$ is enlarged; this is what shows the bottom plot of the figure, in which points associated to subsequent intersections of the unstable manifold with $x = 0.0$ can also be detected.

Refinement of the mesh and additional improvements

We have just seen that if the radius of the grid is not very small only a reduced set of points of Z is determined, even using a relatively large threshold for the orthogonality condition required in Step 2. To overcome this problem a second iterative method has been implemented to find a covering set of Z. The procedure recalls the two-step algorithm implemented in GAIO (see Dellnitz et al. [DH97]) to find covering sets of attractors in dynamical systems. Each iteration of the method consists in two steps: division and selection. In the division procedure the cells of the covering are divided into smaller ones, in the selection step it is verified which of the new cells contain (or can contain) points of Z.

Let R_0 be a rectangle where we look for points of Z, defined by its centre $\vec{c}^{\,0} = (c_1^0, \ldots, c_n^0)$ and its radius $\vec{r}^{\,0} = (r_1^0, \ldots, r_n^0)$), this is:

$$R_0 = \{\vec{x} = (x_1, \ldots, x_n) \mid c_i^0 - r_i^0 \leq x_i \leq c_i^0 + r_i^0, \quad i = 1, \ldots, n\}.$$

Using R_0 as initial rectangle, the two steps of the iterative procedure are:

1. Division: Divide the rectangle in two smaller rectangles with radius (r_j^{i+1}) and centres ($c_{j,\pm}^{i+1}$) given by:

$$r_j^{i+1} = \begin{cases} \dfrac{r_j^i}{2} & \text{if } i = j \pmod n \\ r_j^i & \text{otherwise} \end{cases}, \quad c_{j,\pm}^{i+1} = \begin{cases} c_j^i \pm r_j^{i+1} & \text{if } i = j \pmod n \\ c_j^i & \text{otherwise} \end{cases}.$$

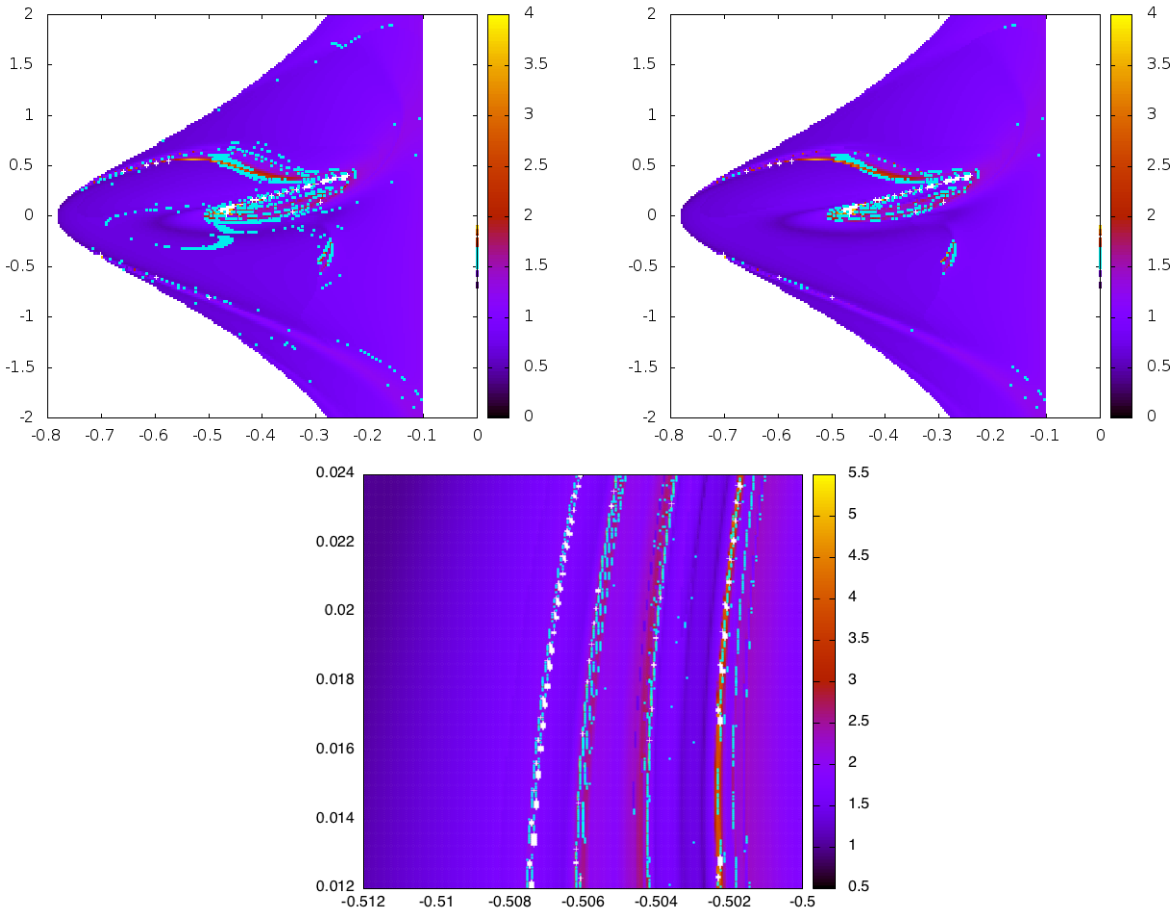


Figure 3.7: Projection of the set $Z_{0.1}$ on the $(y\text{-}\dot{y})$ plane ($x = 0.0$ and $C = 3.1638$) overimposed to the FTLE field. The two plots on the top correspond to a mesh of $200\times$ 200 points. The points of the left-hand side plot fulfil only the second condition of the algorithm with $\delta_1 = 0.1$ and the ones on the right-hand side plot fulfil also the condition of Step 3 ($\lambda_n > 1$). A magnification of this second plot is shown in the bottom plot, using a grid of 200×200 points inside the square $[-0.512, -0.5] \times [0.012, 0.024]$.

2. Selection: In this step one must determine wether the divided rectangles contain or not points of Z. The selection criteria varies along the iterative procedure. In the first selection steps the required conditions are less strict than in the last ones, according to the following: the rectangle R_j is selected if:

   - there exist at least a point $x \in R_j$ such that it verifies the first condition of the hyperbolic LCS characterisation, for values of j less than a fixed given one,
   - there exist at least a point $x \in R_j$ such that it verifies the first and third conditions of the hyperbolic LCS characterisation, for values of j greater than a fixed given one.

   Since it is not possible to verify the above conditions for all the points in R_j, the selection tests are done for the points of a grid defined inside the rectangle. For the points of the grid, and in order to decide if they belong or not to Z, the non-refined test previously explained is applied.

These two selection criteria are introduced in order to capture as many points as possible of the hyperbolic LCS. It may happen that, if the grid is not fine enough to detect the orthogonality condition, the procedure

discards some rectangles containing points in Z. To avoid this situation it is convenient to introduce, aside from the selected and discarded rectangles, the *conditionally-selected rectangles*. A rectangle is said to be conditionally-selected if it does not verify the selection condition but it is beside one verifying it. Then, the selection criteria applies to it if:

- it is contained in a selected or conditionally-selected rectangle of the previous iteration, or
- it is beside a selected rectangle of the current iteration.

In this way, at the end of each iterate of the refinement procedure, the selected rectangles will be surrounded by conditionally-selected ones.

As an additional remark, and due to the great amount of numerical integrations required by the procedure, is important to use a fast numerical integrator. In the examples that follow, a Taylor method has been chosen (see Jorba et al. [JZ05]), which is both accurate and fast for the differential equations considered.

Again for the RTBP model and the same values of the parameters used in the preceding computations, Figure 3.8 shows the resulting selected points, verifying the the orthogonality condition and $\lambda_n > 1$ (cyan), after the iterations number 8 and 10 of the division/selection procedure. The procedure has been applied starting with a grid of 200×200 points and the selection criteria has been applied using inside each rectangle a grid with 21×21 points.

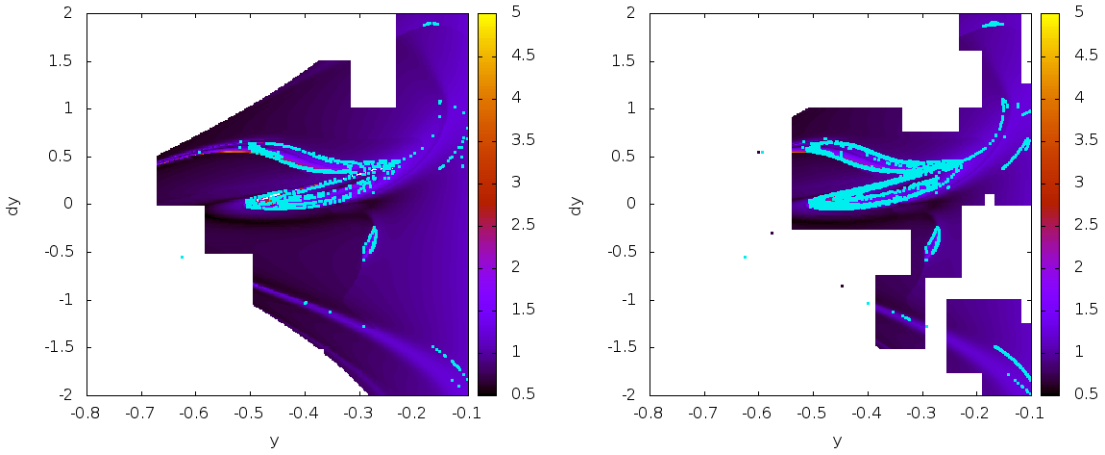


Figure 3.8: For $C(x, y, \dot{x}, \dot{y}) = 3.1638$ and $x = 0$, $(y, \dot{y})$ apoapsis-FTLE field and selected points in $Z_{0.1}$ verifying $\lambda_n > 1$ (in cyan) determined using the division/selection algorithm, up to iteration number 8 (approx. 321×321 points)(left) and 10 (approx. 641×641 points) (right).

Using the non-refined test, a set of points, with a similar degree of accuracy as the one obtained after the 10th iteration, can be computed using a grid with 641×641 points. Table 3.1 shows the CPU time improvement obtained with the usage of the division/selection algorithm. Observe that for small grids this algorithm does not produce a big save of CPU time compared with the non-refined one. However, when the dimension of the grid increases, the division/selection algorithm becomes faster reducing the CPU time by more than a quarter. The time improvements has sense since the first iterates do not discard cells. However, the deeper the algorithm goes, the more boxes are discarded, and the more efficient the algorithm is.

Another fact to take into account is the initial value of the parameter δ. When this parameter is small one gets very accurate determinations of Z_δ but then very large grids are required. This is shown in Figure 3.9, where the sets $Z_{0.1}$ and $Z_{0.02}$ obtained after the 10th iterate of the division/selection algorithm are displayed. It can be seen that $Z_{0.1}$ almost covers all the first intersection of the invariant manifold of the

Size of the grid	CPU time using division/selection	CPU time without division/selection
200×200	86 s	103 s
643×643	253 s	1087s

Table 3.1: Comparison between the CPU time used by the non-refined and the division/selection procedures for different grid dimensions and final time $T = -6$.

periodic orbit, while for $Z_{0.02}$ there are some regions of the invariant manifolds that are not covered. The two plots of this figure have been computed using a grid with the same number of points, so if $\delta_2 < \delta_1$ then the number of points detected in Z_{δ_2} will be less than the ones detected in Z_{δ_1}. Note also that the other intersections of the manifold shown in the image (the second and the third ones) are only partially covered by the set Z_δ, due to the fact that the FTLE are computed up to a final time T not large enough to capture them. Using a larger final time, more intersections would be covered by Z_δ.

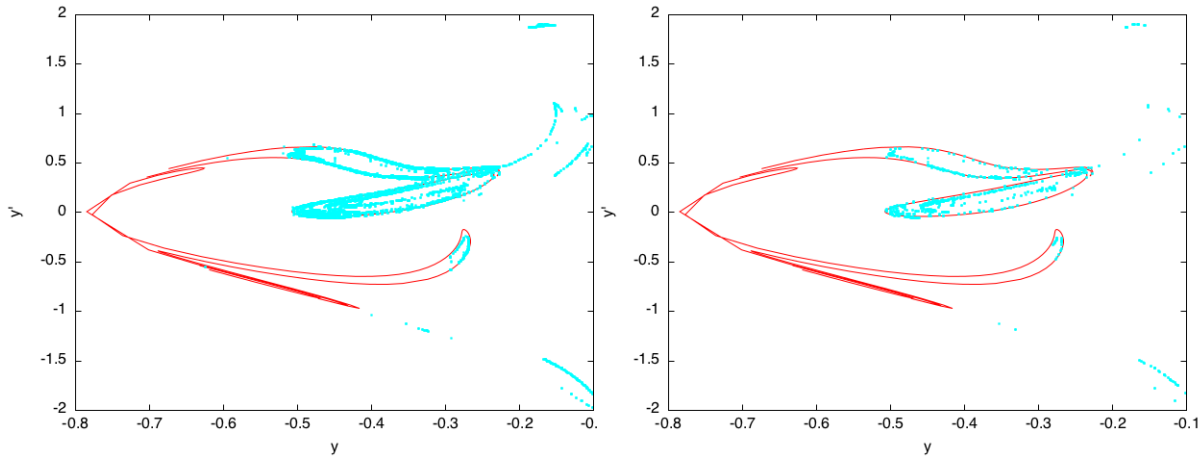


Figure 3.9: In cyan, the sets $Z_{0.1}$ (left) and $Z_{0.02}$ (right) verifying the condition $\lambda_n(\vec{x}) > 1$. The computations have been done using the division/selection algorithm up to the iterate number 10. The first three intersections of the unstable invariant manifold associated to L_1 (red) are also represented in both figures.

In order to refine more the determination of Z_δ, one can try to require the scalar product condition for different final times and then take the intersection of the resulting Z sets. The reason for this is based on the fact that the points in the intersection of the invariant manifolds with the Poincaré section should verify all the conditions of Theorem 3 for a given T large enough. From that time on, all the possible times should be illegible for the criterion, therefore we can look for those points that are in Z_δ for a number of given epochs $T_1 < T_2 < T_3 < \cdots < T_n$ (being T_1 large enough).

For instance we can look to those sets such that verify:

$$\begin{aligned} |\left\langle \nabla\lambda_n(\vec{x}_0, t_0, T_1), \vec{\xi}_n(\vec{x}_0, t_0, T_1) \right\rangle| &< \delta \\ |\left\langle \nabla\lambda_n(\vec{x}_0, t_0, T_2), \vec{\xi}_n(\vec{x}_0, t_0, T_2) \right\rangle| &< \delta \\ |\left\langle \nabla\lambda_n(\vec{x}_0, t_0, T_3), \vec{\xi}_n(\vec{x}_0, t_0, T_3) \right\rangle| &< \delta \end{aligned}$$

being $T_1 = 8, T_2 = 9$ and $T_3 = 10$. Figure 3.10 shows those points that fulfil the condition for T_1 (black), T_2 and T_3 (red), and those ones that verify the three conditions simultaneously (yellow). There, one can see

that there are spurious points in each of the three sets, but when we look to their intersection most of the spurious points disappear.

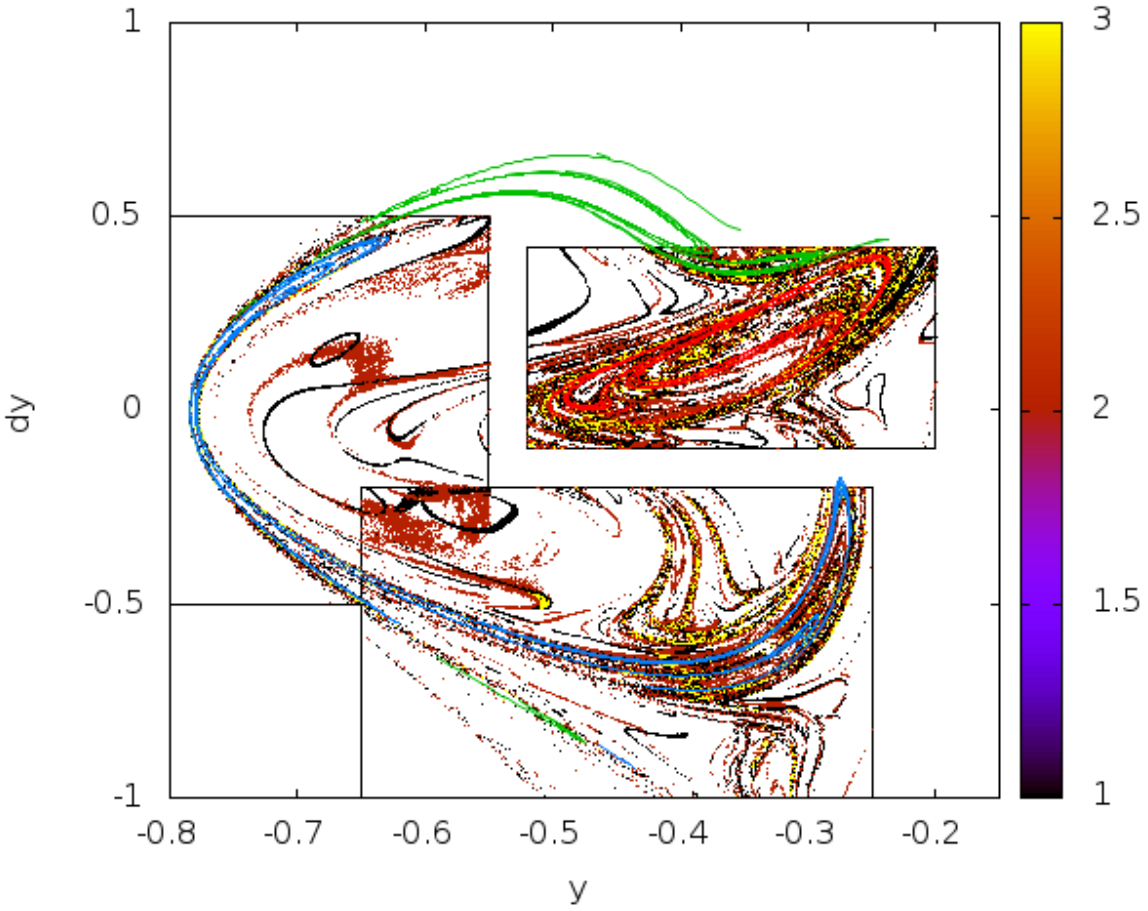


Figure 3.10: The sets of points in $Z_{0.005}$ after 20 iterates of the division/selection algorithm for time $T_1 = 8$ (black), $T_2 = 9$ and $T_3 = 10$ (red), and T_1, T_2 and T_3 (yellow). The first (red), second (green) and third (blue) intersection of the invariant manifold with the plane $x_0 = 0$ is also displayed in the figure.

Figure 3.11 is an enlargement of the first intersection of the invariant manifold together with the set of points that are in $Z_{0.005}$ after 20 iterates of the algorithm and fulfilling the three epoch conditions simultaneously. There, one can see that the invariant manifold is not completely detected. Furthermore, the are still some spurious points detected by the algorithm that are not in the manifold.

As a first conclusion, one can say that the hyperbolic LCS are useful to detect the proximity of invariant manifolds, but are not enough accurate for its precise determination. This is mainly due to the inaccuracy of the numerical computations around the invariant manifolds, due to the fact that in those neighbourhoods the large hyperbolicity effects make difficult to do the computations with enough precision.

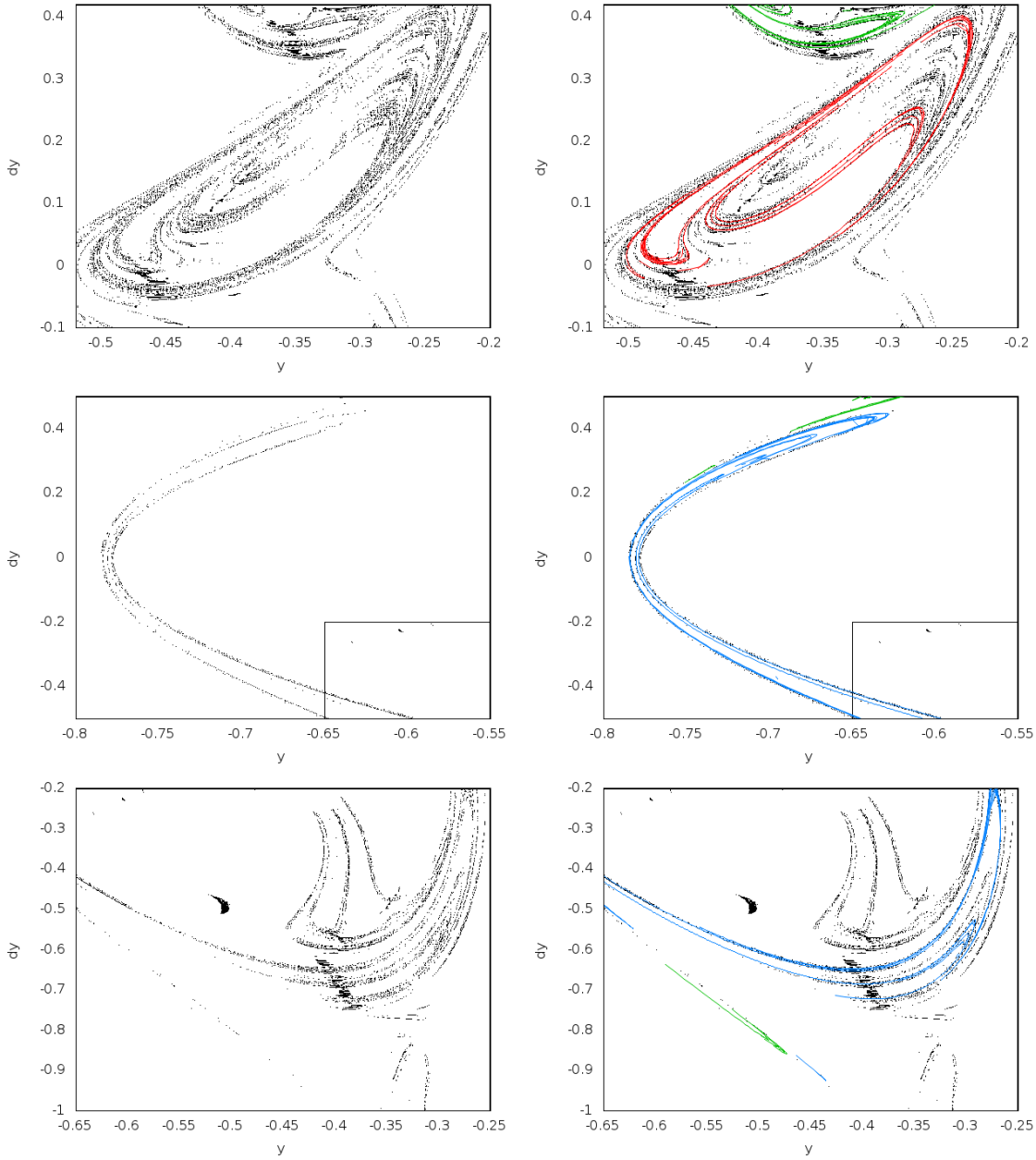


Figure 3.11: From top to bottom, the set of points in $Z_{0.005}$ after 20 iterates of the algorithm for $T_1 = 8$, $T_2 = 9$ or $T_3 = 10$ (left) around the first, second and third intersections of the manifold respectively. The same plot with the first (red), second (green) and third (blue) intersections of the invariant manifold are shown in the right-hand side plots.

3.2.3 Some tests on the approximation of invariant manifolds using hyperbolic LCS

If the invariant manifolds of the systems are known, for instance in the pendulum or the CRTBP, we can check if the conditions required in Algorithm 3 are verified or not. In particular, we are interested in checking the scalar product condition $\left\langle \nabla\lambda_n(\vec{x}_0,t_0,T), \vec{\xi}_n(\vec{x}_0,t_0,T) \right\rangle = 0$ on the manifold.

Figure 3.12 shows the values of the scalar product for the points of the first intersection of the unstable invariant manifold of the periodic orbit around L_1 with the hyperplane $x = 0$. The orbits of the unstable manifold have been parametrised by an index i related to the angle along the periodic orbit where their initial conditions have been computed. These initial conditions have been integrated up to a final time equal to T=–2, –5 and –7 a-dimensional time units. The values of $\nabla\lambda_n$ are then approximated using numerical derivatives. In this figure one should expect for the scalar product to be be zero for all the points $\vec{x}_0$ in the intersection of the invariant manifold with $x = 0.0$, however this is not the case. For a small value of T the product is far from zero. This can be explained considering that after this time interval the propagated points have not yet reached the periodic orbit and still behave in a very regular way. However, as the time interval increases, the indicator gets closer to zero, except for a decreasing (with $|T|$) set of orbits of the manifold. As a consequence, the set Z_δ used in Algorithm 3 appears not to be the most suitable one for the accurate determination of invariant hyperbolic manifolds.

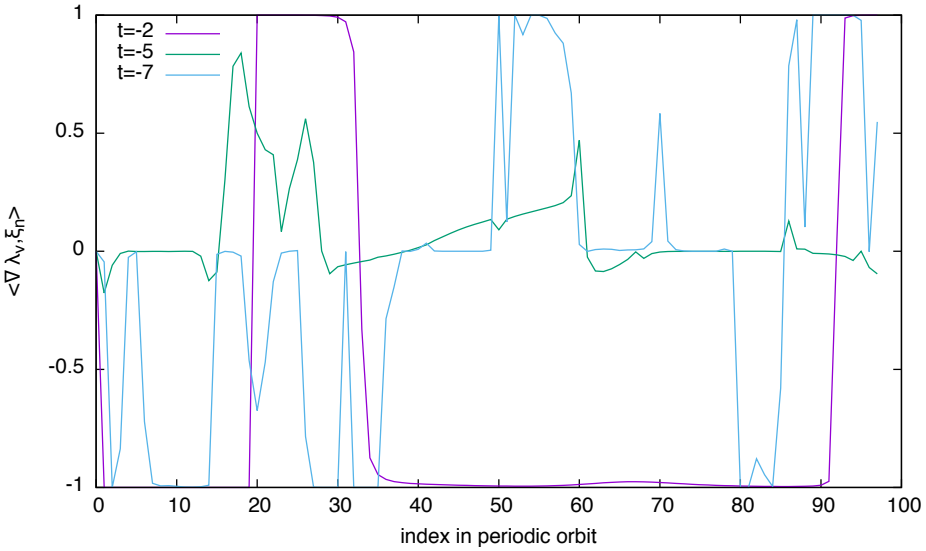


Figure 3.12: For different values of $T = -t = 2, 5, 7$, value of the scalar product $\left\langle \nabla\lambda_n(\vec{x}_0,t_0,T), \vec{\xi}_n(\vec{x}_0,t_0,T) \right\rangle$ for points $\vec{x}_0$ in the intersection with $x = 0.0$ of the invariant manifold of the periodic orbit around L_1 of the Earth-Moon RTBP with $C_j = 3.61$. The points are parametrised by an index on the periodic orbit.

Figure 3.12 shows that the set Z_δ as described in algorithm 3 is not working well enough. For systems with a first integral H, as the one under consideration, some additional tests have been done using the alternative condition given by 3.2.

In order to compute the new scalar product $\left\langle \nabla\rho_{t_0}^{t_0+T}(\vec{x}_0), \nabla H(\vec{x}_0,t_0) \right\rangle$ one must first compute the value of:

$$\rho_{t_0}^{t_0+T}(\vec{x}_0) = \frac{\|\nabla H(\vec{x}_0,t_0)\|}{\sqrt{\left\langle \nabla H(\vec{x}_0,t_0), \left(C_{t_0}^{t_0+T}(\vec{x}_0)\right)^{-1} \nabla H(\vec{x}_0,t_0) \right\rangle}},$$

This computation can be done as follows:

1. Denoting by C the Cauchy-Green tensor $C_{t_0}^{t_0+T}(\vec{x}_0) = J^*J$, where $J = D\phi_{t_0}^{t_0+T}(\vec{x}_0)$, compute the product $C^{-1}\nabla H = \vec{z}$.

   Using the factorisation $J = J_n \dots J_2 J_1$, the product $\vec{z}$ can be computed recursively solving:

$$C\vec{z} = \nabla H \quad \Leftrightarrow \quad J_1^* \underbrace{J_2^* \dots J_n^* J_n \dots J_2 J_1 \vec{z}}_{\vec{z}_{1*}} = \nabla H$$

   In this way, the first system to be solved is $J_1^*\vec{z}_{1*} = \nabla H$, the second is $J_2^*\vec{z}_{2*} = \vec{z}_{1*}$. After $2n$ steps of the procedure one gets the value of $\vec{z}$. At each step, it is convenient to normalise the solution in order to minimise the error.

2. Compute the denominator of $\rho_{t_0}^{t_0+T}(\vec{x}_0)$. Since

$$\begin{aligned}
\left\langle \nabla H(x_0,t_0), \left(C_{t_0}^{t_0+T}(x_0)\right)^{-1} \nabla H(x_0,t_0) \right\rangle &= \nabla H^* \left(J^*J\right)^{-1} \nabla H \\
&= \nabla H^* J^{-1} (J^*)^{-1} \nabla H \\
&= \left((J^{-1})^* \nabla H\right)^* \left((J^{-1})^* \nabla H\right) \\
&= \left\langle (J^{-1})^* \nabla H, (J^{-1})^* \nabla H \right\rangle \\
&= \|(J^{-1})^* \nabla H\|^2,
\end{aligned}$$

   the denominator is equal to $\|(J^{-1})^*\nabla H\|$. The recursive procedure of the previous step can also be used to compute $(J^{-1})^*\nabla H$ without inverting the matrix J and solving the system $\nabla H = J^*\vec{z}$ instead.

3. Instead of computing ρ using the previous formula, it can also be computed, according to its definition, as:

$$\rho_{t_0}^{t_0+T}(\vec{x}_0, \vec{n}_0) = \left\langle \vec{n}_T, D\phi_{t_0}^{t_0+T}(\vec{x}_0)\vec{n}_0 \right\rangle = \langle \vec{n}_T, J\vec{n}_0 \rangle = \frac{1}{\|\nabla H_T\| \|\nabla H_0\|} \langle \nabla H_T, J \nabla H_0 \rangle$$

   In this case one can compute the matrix J in the same way as in the previous cases or we can propagate the gradient through the field using the variational equations. Now instead of using a $n \times n + n$ system of differential equations, with the identity as initial condition, the system will be of only $2n$ equations and with the gradient of the energy ∇H as initial condition for the variational part.

   An important advantage using this method is that we can normalise the vector $J\nabla H$ whenever its norm is bigger or smaller than a certain tolerance $\varepsilon_{\max}$ and $\varepsilon_{\min}$.

Figure 3.13 shows the value of the scalar product 3.2 as a function of the index associated to the orbits of the unstable manifold. Again, the values of this product are far from zero. Similar results are obtained for the pendulum.

For the pendulum, Figure 3.14 shows the behaviour of $\nabla\rho$, in a small region around the invariant manifold of the equilibrium point (in green), together with the value of ρ (in colours). There, one can see that the invariant manifold is in the region with large values of ρ. In addition, at each side of the invariant manifold the gradient of ρ points in opposite directions; therefore, since the gradient is continuous, there must be some point where the gradient of ρ is orthogonal to the gradient of the energy. Remember that the gradient of the energy of the pendulum is orthogonal to the hyperbolic manifold.

According to the continuity of the scalar product condition and using a continuation method, one can look for the set of points verifying:

$$\left\langle \nabla \rho_{t_0}^{t_0+T}(\vec{x}_0), \nabla H(\vec{x}_0, t_0) \right\rangle = 0.$$

Figure 3.15 shows the results of the above computation for several final times T of integration. It is clearly seen that the longer the integration time T the better is the invariant manifold approximated. However, there are some points where the invariant manifold is not approximated at all. In the same figure the logarithm of the distance to the separatrix is shown. As we claimed, the left part of the points which are close to

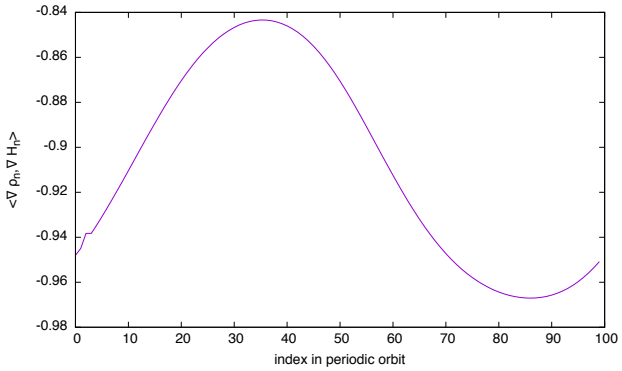


Figure 3.13: For $T = -5$ time units, value of the scalar product $\left\langle \nabla \rho_{t_0}^{t_0+T}(\vec{x}_0), \nabla H(\vec{x}_0, t_0) \right\rangle$ for points $\vec{x}_0$ in the intersection with $x = 0.0$ of the invariant manifold of the periodic orbit around L_1 of the Earth-Moon RTBP with $C_j = 3.61$. The points are parametrised by an index on the periodic orbit.

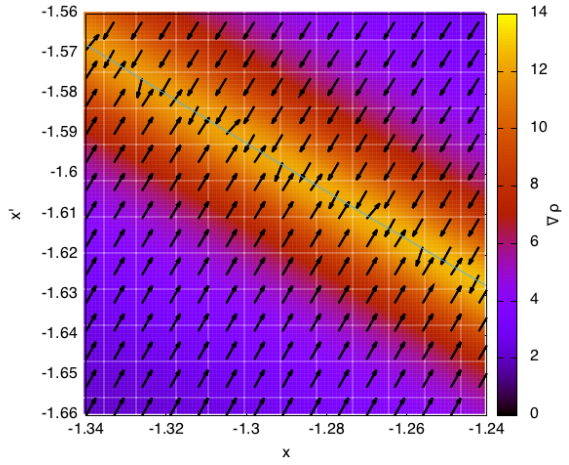


Figure 3.14: For the pendulum, value of $\rho_{t_0}^{t_0+T}$ in a region around the separatrix (in green). The arrows indicate the direction of the gradient of ρ.

the hyperbolic stable invariant manifold are well computed but the ones on the right close to the unstable manifold are not.

The above examples show that the hyperbolic LCS can be useful to give a first approximation of the hyperbolic invariant manifolds but not for a precise determination of them. This gives a motivation to explore new indicators to detect invariant manifolds. The Jet Transport techniques, introduced in the previous chapter, are a good option to explore high order alternatives. Observe that the LCS are computed using only the state transition matrix, which is the first order deviation of the flow. With the Jet Transport higher order terms can be introduced in the procedures that may improve the results. This is what will be done in the following sections.

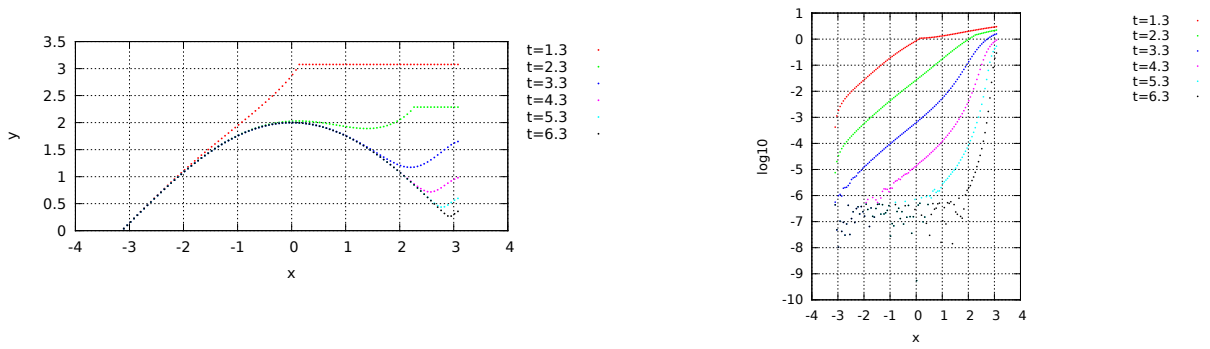


Figure 3.15: For the pendulum, and for different final times T, set of points that fulfil the scalar product condition $\left\langle \nabla \rho_{t_0}^{t_0+T}(\vec{x}_0), \nabla H(\vec{x}_0, t_0) \right\rangle = 0$ (left plot). The right-hand side plot displays the logarithm of the distance between of these points and the invariant manifold.

3.3 Non-linear alternatives for the computation of LCS

In the previous sections, the efficiency of the hyperbolic LCS (according to the definition given by G. Haller) for the approximation of invariant manifolds has been discussed. In what follows we will make use of the Jet Transport techniques to define other dynamical indicators that can be used to determine the presence of hyperbolic structures.

Four different indicators will be introduced, that make use of the high order variational terms provided by the Jet Transport. The first indicator uses the maximum initial neighbourhood that can be used by the Jet Transport ensuring a certain accuracy. The next two ones are based on the detection of the maximum expansion and contraction directions and the associated ratios of expansion and contraction. The fourth one is a small variation of these last two, but the expansions are controlled in the normal spaces to the orbit instead of in the full phase space.

In all the numerical computations that follow, the order of the Taylor method used for the numerical integration has been set equal to 25, and the order n of the jet expansions has been taken equal to 5 for the pendulums and equal to 2 for the restricted three body problems.

In principle, it could be enough to use a jet of order 1 to detect the hyperbolic structures (with less CPU effort) and, in fact, all the algorithms that follow can be used with any order of the jet. However, the procedures that require the determination of expansion and contraction rates, need two consecutive orders of the jet for the computation of the derivatives, so, in these two cases, at least order 2 must be used.

3.3.1 Maximal initial boxes

If a point is close to an hyperbolic structure, after some time the points in its neighbourhood can behave in different ways and, in general, they will diverge from each other. As a result, the final box (image of the initial neighbourhood) will increase its size, making impossible to approximate the position of neighbouring points, within a certain accuracy, using a polynomial of a given degree (maybe except if the size of the initial box is small enough). On the contrary, if the point is far away enough from an hyperbolic structure, the surrounding points will, in general, behave in a more regular way and it will possible to use bigger initial boxes.

Figure 3.16 exemplifies the previous comment taking the pendulum as an example. It shows the final box (green) for three different small rectangles centred at at $(2, 0)$, $(2.5, 0)$ and $(3, 0)$, respectively. In the three cases the initial box considered has a size of 0.01×0.01 and is propagated up to order 9. The figure shows also the boundaries of the farthest points from the centre that in principle must enclose the propagated box (grey). We can see that for those points that are far away from the manifold the propagation of the box remains inside the boundary lines. However, the last box, which is close to the separatrix, has diverging edges. This happens because the polynomial that has been used is not good enough to represent the image of the initial box, mainly due to the presence of an hyperbolic structure.

According to the above remark, the first indicator considered is the size $\xi_{\max}$ of the maximum initial box that can be taken such that the propagated images fulfil the following ε_{jet} precision requirement:

$$\xi_{\max} = \min_{|k|=n} \left(\frac{\varepsilon_{\text{jet}}}{a_{m,k}} \right)^{1/k},$$

where $a_{m,k}$ is the coefficient of the Taylor approximation of the m-th component of the solution obtained by the jet $\phi_m(t; \vec{x}_0 + \vec{\xi}, t_0) = \sum_{|k|\leq n} a_{m,k} \vec{\xi}^k$.

Figure 3.17 shows the maximum size of the initial boxes to maintain a given precision $\varepsilon_{jet} = 10^{-6}$ for a grid of points for the simple pendulum as well as for the CR3BP and the ER3BP. As it can be seen, the points that are close to an invariant manifold have lower values than all those that are far away. The basic structures detected agree well with the ones determined using FTLE and displayed in Figure 3.2.

Figure 3.18 shows the difference between the maximum box sizes, $\xi_{\max}$, computed using the jets of order 1 and 5. It can be seen that near the hyperbolic structures the improvement for the size of the box is small (of the order of 10^{-5}) while close to the origin the improvement is increased almost up to 0.1. This means that the use of a high order jet allows the detection of the regular regions with less computational effort.

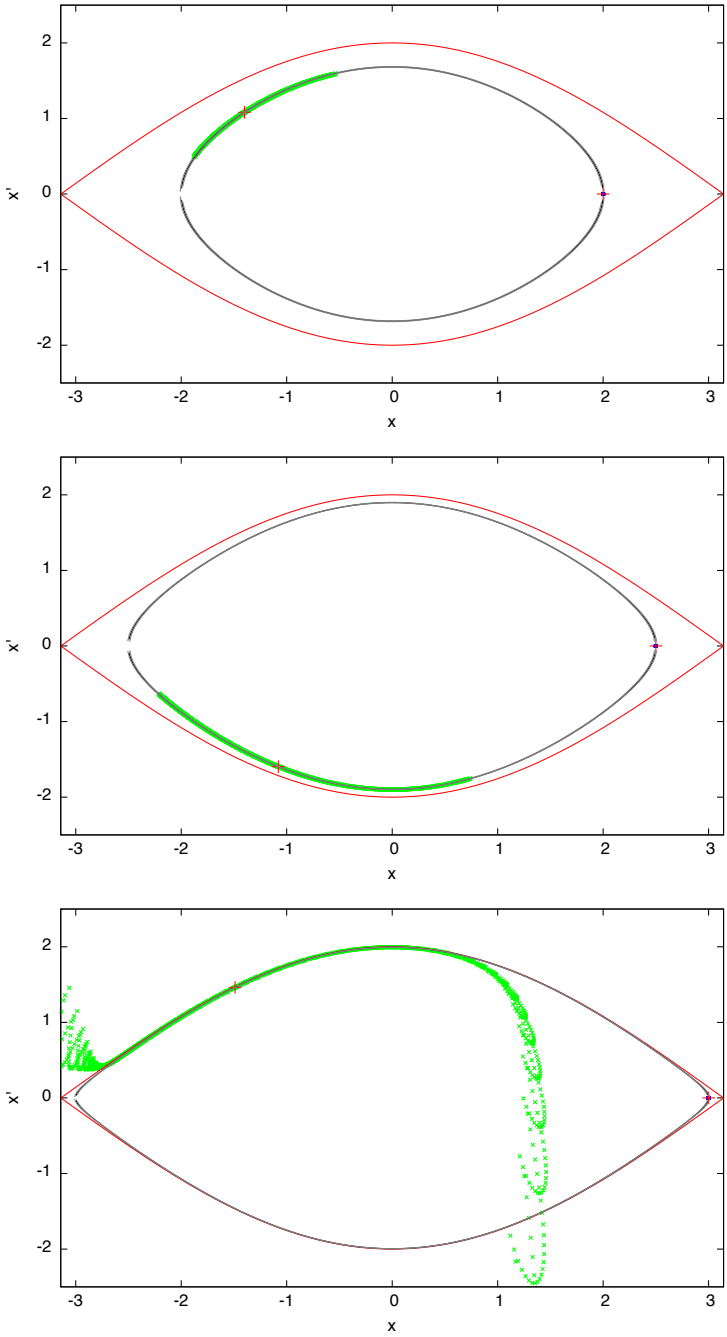


Figure 3.16: From top to bottom, images (green) of an initial small box (blue) centred at (2,0), (2.5,0) and (3,0), respectively, of size 0.01×0.01 after 189 time units. The computations have been done using variational terms up to order 9. The orbit of the initial point is displayed in black. The grey curves are the inner and outer orbits of the points in the initial box.

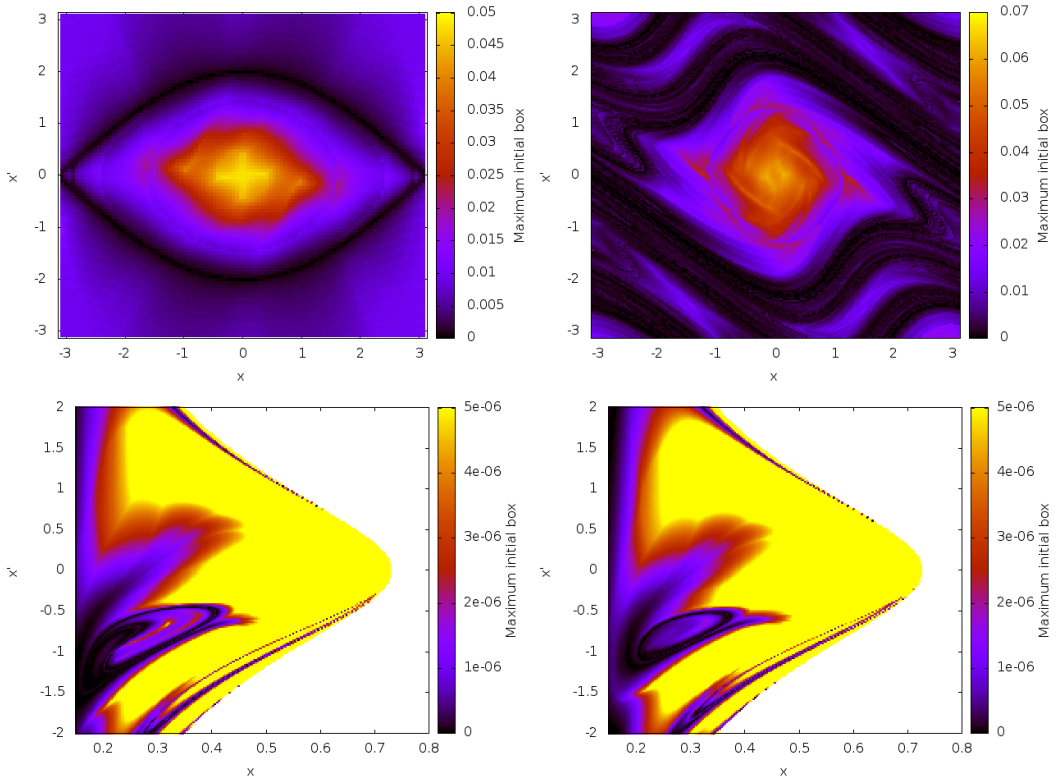


Figure 3.17: Size $\xi_{\max}$ of the maximum initial boxes to maintain a given precision $\varepsilon_{jet} = 10^{-6}$ for the simple and perturbed pendulums (top) and for the CR3BP and ER3BP (bottom). In the pendulum case, the computations have been done using a final time around $T = 10$. The two bottom figures have been computed with a fixed final time $T = 2$ adimensional units. In the two last examples, the mass parameter is $\mu = 0.1$, the values of the Jacobi constant used to reduce dimension of the problem are $E_0 = -1.8$ and $\tilde{E}_0 = -1.8$ and the hyper-plane condition is $y = 0$. The initial phase for the elliptic problem is $f_0 = 0$. This means that the grid used for the computation of the initial conditions is of the form: $(x_i, 0, \dot{x}_j, \dot{y}(x_i, 0, \dot{x}_j, E_0) > 0)$.

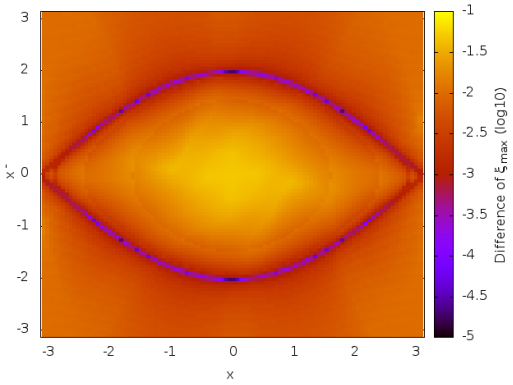


Figure 3.18: Difference between the two sizes, $\xi_{\max}$, determined using jets of order 1 and 5, and a precision $\varepsilon_{jet} = 10^{-6}$, for a pendulum with a final integration condition time close to $T = 30$.

3.3.2 Expansion and contraction measures

Using the same basic idea introduced in [Hal11] for the determination of the hyperbolic LCS, one can obtain similar results computing the maximum expansion and maximum contraction ratios from the Jet Transport. This is what is considered and discussed in this section.

In the 1-degree of freedom case, one can consider a circle of radius r_0 around an initial condition $\vec{x}_0$ and, on this circle, look for the angle θ_+ associated to the most expanding direction as well as to the one corresponding to the most contracting one θ_-, i.e. the angles such that the difference $\|\phi(0;T,\vec{x}_0) - \phi(0;T,\vec{x}_0+\vec{\xi}(\theta)\|$ is maximum and minimum, respectively, and subjected to $\|\vec{\xi}(\theta)\| = r_0$. Once the results of the Jet Transport are available, instead of computing the images of the points on the circle by numerical integration, one can obtain the values of $\phi(0;T,\vec{x}_0+\vec{\xi}(\theta))$ in an easy and fast way evaluating $P_T(\vec{\xi}(\theta))$. The procedure proceeds as follows:

1. Take an initial state $\vec{x}_0$.
2. Propagate the state together with its jet up to a certain time T. In this way the polynomial $P_{T,\vec{x}_0}(\vec{\xi})$ approximating $\phi(0;T,\vec{x}_0+\vec{\xi})$ is determined.
3. Compute the maximum initial box and define its radius as r_0.
4. Using $P_{T,\vec{x}_0}(\vec{\xi})$, compute the images of a first coarse grid of initial conditions, on the circle of radius r_0, parametrised by an angle θ:

   $$(x_0,\dot{x}_0) + \vec{\xi}(\theta) = (x_0,\dot{x}_0) + (r_0\cos\theta, r_0\sin\theta).$$

   Determine a first approximation of the angles giving the maximum expansion, $l_+ = \max_\theta \|P_{T,\vec{x}_0}(\vec{\xi}(\theta)) - P_{T,\vec{x}_0}(0)\|$ and the maximum contraction, $l_- = \min_\theta \|P_{T,\vec{x}_0}(\vec{\xi}(\theta)) - P_{T,\vec{x}_0}(0)\|$, with $\|\vec{\xi}(\theta)\| = r_0$.
5. Use the initial approximations obtained in the previous step to refine the values of $l_\pm$ using a maximum/minimum search algorithm. The computations are done by means of Newton's method together with the results of the Jet for the required derivatives.
6. Compute the expansion and the contraction rates, $\lambda_\pm = l_\pm/r_0$.

In the last step, the expansion/contraction rates are divided by the initial size of the box. This is done in order to normalise the expansions and contractions. If the initial box used is bigger, the maximum size of the final box will also be bigger. Taking the rate of contraction allows the comparison of different initial conditions.

The computation of the distance $\|P_{T,\vec{x}_0}(\vec{\xi}(\theta)) - P_{T,x_0}(0)\|$ requires some additional comments. If the initial box has a relatively big maximum size it may happen that the final box has a banana shape. Figure 3.19 (top, right) shows an example of this situation. In some cases, the Euclidean distance between the farthest point of the boundary of the final box to the image of the centre of the initial box is computed as the length of a segment that is not contained in the final box, and this is not a good estimate of the distance between those two points. For this reason, two different ways of computing the value $l_\pm$ have been considered:

- The first method computes $l_\pm$ with the usual Euclidean distance, even if doing so we are going through the exterior of the final box.
- The second method considers the segment that joints the preimage of the most expansive/contractive point $\vec{\xi}(\theta_+)$ and $\vec{\xi}(\theta_-)$ with the centre $\vec{x}_0$ of the initial box, and determines the length of the image of this segment through the flow. The length of this segment, that it is parametrised using N points as $\vec{x}_i = \vec{x}_0 + \frac{i}{N}(\vec{\xi}(\theta_\pm) - \vec{x}_0)$, is defined as the length of the polygonal that joins the images $P(\vec{x}_i)$.

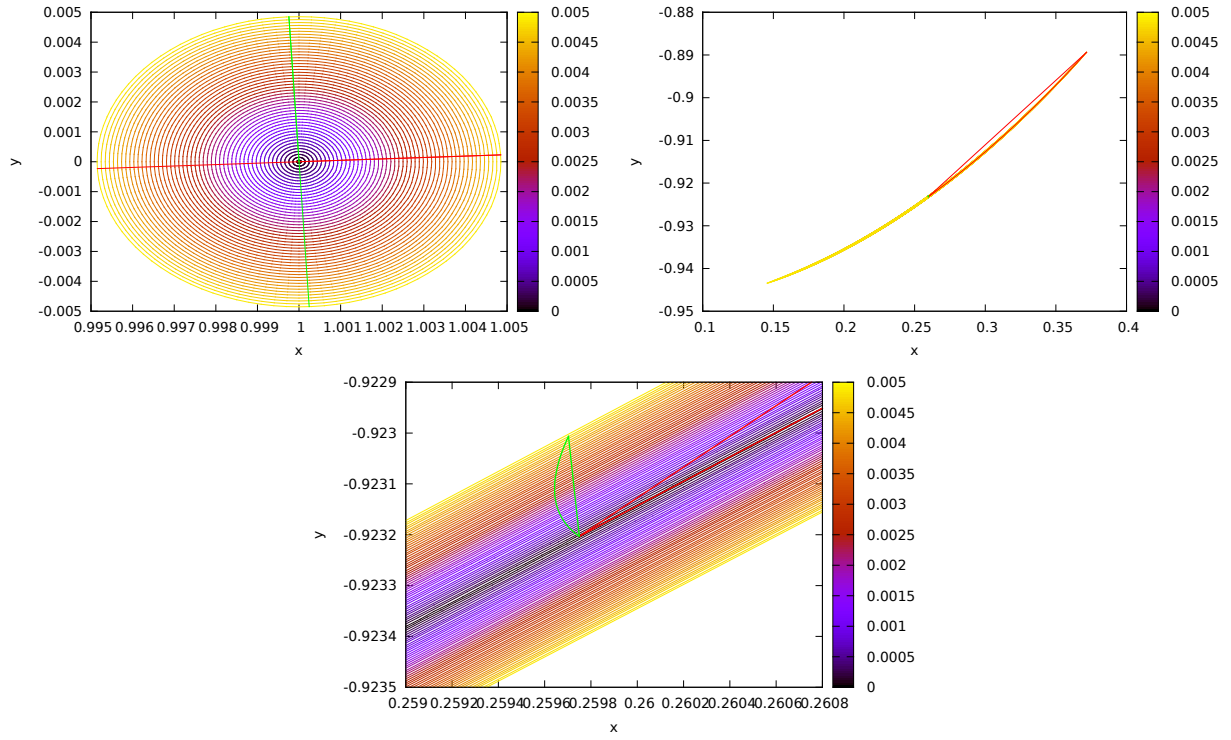


Figure 3.19: For the pendulum: initial box around the point (1,0) (top left); image, after $T = 181899$ time units, of the initial box (top right); magnification of the image of the initial box (bottom). In the three figures, the colorbar denotes the Euclidean distance from every point in the box to the initial condition. The red and green lines in the first plot are the segments that joint the preimages of the most expansive/contractive points to the centre $\vec{x}_0$ of the box. The images under the flow of both segments are represented in the other two figures. The two green curves of the bottom figure are the ones used to compute l_- according to the two procedures explained in the text.

In order to differentiate which distance has been used, the subindex 1 is added when the Euclidean distance is used and the subindex 2 in the other case. Therefore, two different expansion ($\lambda_{+,1|2}$) and contraction ($\lambda_{-,1|2}$) rates will be computed. Figure 3.19 illustrates the two different ways to compute the maximum distance (red) and the smaller distance l_- (green). The first procedure uses the straight segment giving $l_{\pm,1}$, and the second method uses the other curve and gives $l_{\pm,2}$.

Table 3.2 shows the results for the three dynamical indicators introduced up to now, for the unperturbed pendulum, using different initial conditions and box sizes. Observe that if the initial condition is not changed, then the maximum initial box $\xi_{\max}$ does not change with the size of the initial box. On the other side, the values obtained for the expansions and contraction rates are almost independent of the initial distance, this is due to the normalisation done for both rates. Another fact that can be observed is that the differences between the distances using both methods give similar results. As it should be expected, always the l_2 distance is larger than the l_1. As a final remark, observe that although all the results give similar results using the same initial condition, the one using the distance l_2 gives more relative differences as a function of the initial box size. Therefore, despite of it seems that has more sense, the usual Euclidean distance will be preferred for the computations.

The results obtained with the above procedure (using $\lambda_{\pm,1}$), for the simple and perturbed pendulums, are displayed in Figure 3.20 and Figure 3.21 respectively. Both figures also show the differences obtained using the two different options for the final time: a fixed final time and a close approach final time. As it

i.c.	box size	$\lambda_{+,1}$	$\lambda_{-,1}$	$\lambda_{+,2}$	$\lambda_{-,2}$	ξmax
(1,0)	0.007	24.13	0.0416	24.15	0.0669	2.08-02
(1,0)	0.005	24.10	0.0416	24.11	0.0564	2.08-02
(1,0)	0.0007	24.02	0.0416	24.02	0.0420	2.08-02
(2,0)	0.007	96.59	0.0108	96.72	0.1237	8.48-03
(2,0)	0.005	97.83	0.0108	98.10	0.1726	8.48-03

Table 3.2: Hyperbolicity indicators for the unperturbed pendulum computed after 189 time units. in an unperturbed pendulum for different initial condition (i.c.) and initial box sizes.

happens with the FTLE, using a close approach final time gives a notable improvement for the unperturbed pendulum. However the improvement in the perturbed one is not so big. Note that, in all the cases, those zones with big expansion or contraction rates coincide with the ones with high FTLE shown in Figure 3.2, so both the expansion and contraction rates can also be used for the detection of the hyperbolic LCS separating structures.

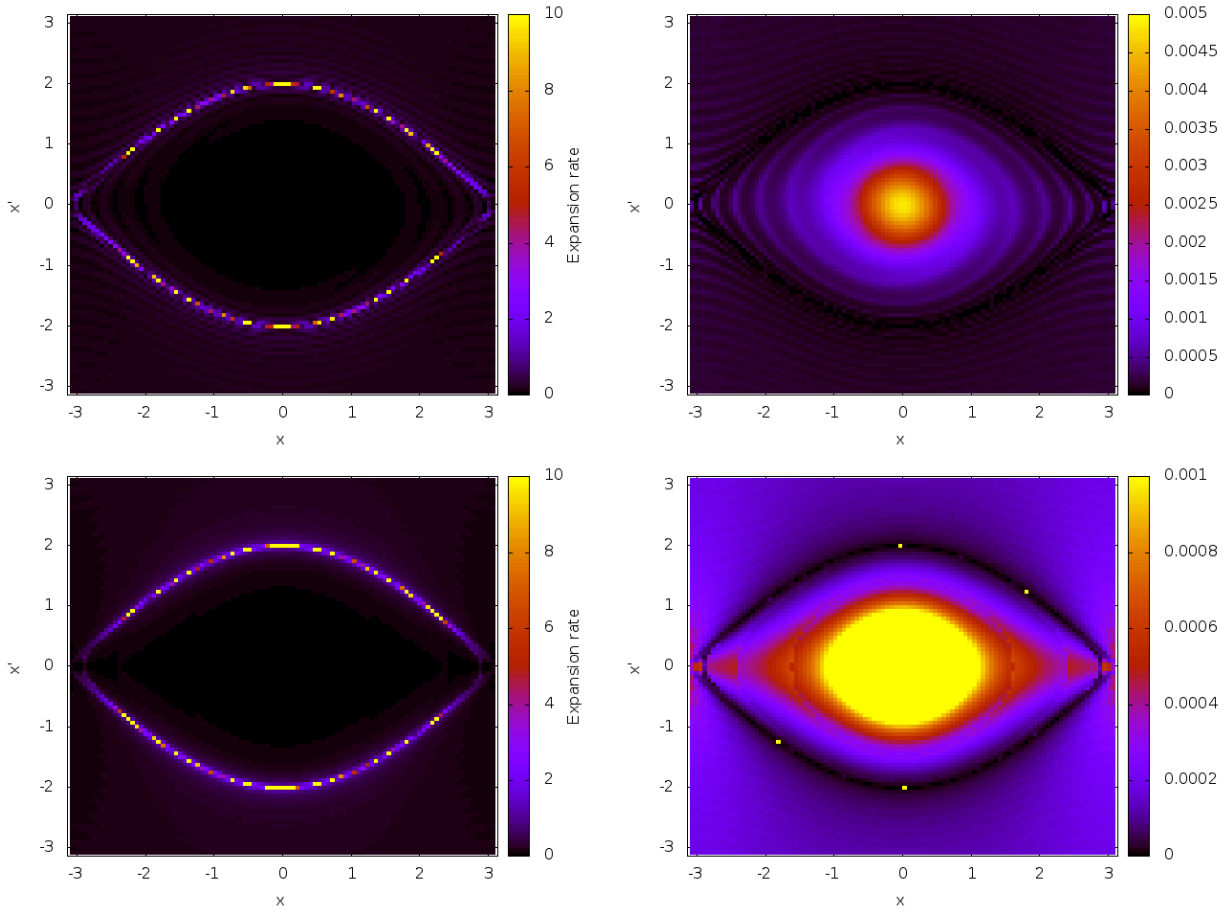


Figure 3.20: For the simple pendulum, maximum expansion rates (left) and maximum contraction rates (right). With a final time of $T = 30$ time units (top) and a final time following the close approach criterion close to a final time of $T = 30$ time units (bottom).

Before dealing with higher dimensions let us see how the final time of integration modifies the value of

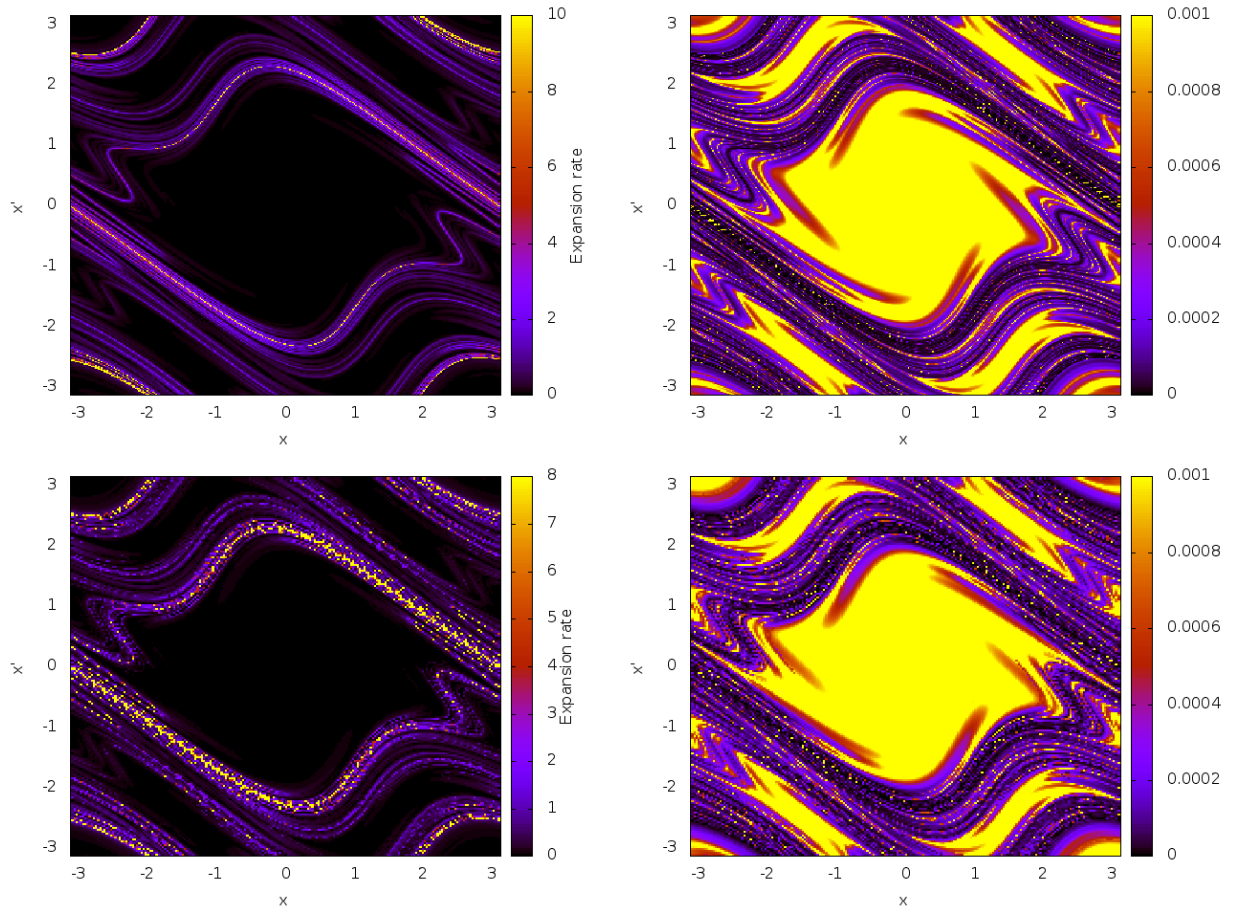


Figure 3.21: For the perturbed pendulum, maximum expansion rates (left) and maximum contraction rates (right). The final time of integration is fixed to $T = 10$ units of time (top) and following the close approach criterion close to a final time of $T = 10$ time units bottom

the expansion and contraction rates. The expansion rate is, somehow, similar to the Lyapunov exponent. Therefore, it has sense to think that the ratio of increasing of the expansion rate λ_+/t will become constant after some transient time. Following the same idea, one should expect that the expansion rate of any point in the orbit will be the same after some long time. However, this is not true.

Figure 3.22 shows the evolution of $\frac{\lambda_{+,1|2}}{t}$ and $\frac{\lambda_{-,1|2}}{t}$ for final times of the form $t = T_t + k \cdot T_p$ and the initial condition $(0, 1.5)$, where T_t denotes the transient time and T_p is the period of the orbit. Using multiples of the period of the orbit the plots become smoother. One can see that when the final time increases the value of expansion rates does not stabilise. On the other hand, the contraction rate stabilises after some transient time interval. This can be due to the fact that the points around the most contractive points are computed with better accuracy than the ones around the most expansive. This may happen because the more expansive point is closer to the value where the Jet does not approximate correctly the neighbour of the initial condition x_0.

Figure 3.23 shows the value of λ_+/T for different final times along the same orbit parametrised by s, defined as the difference between the initial and final time. Using Lyapunov exponents all the values for the same orbit should converge to the same value (since the Lyapunov exponent is constant along orbits). However, here one can see that increasing the final time (that varies between 20 and 170 time units) each

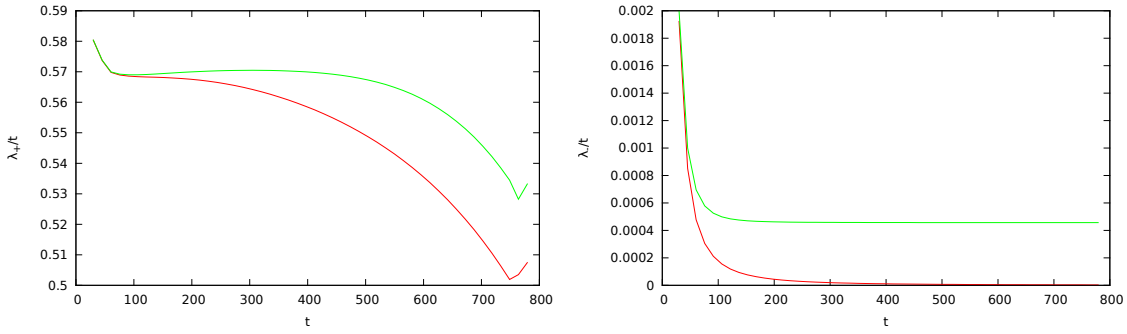


Figure 3.22: Normalised values of $\frac{\lambda_{+,1|2}}{t}$ (left) and $\frac{\lambda_{-,1|2}}{t}$ (right) for $t = T_t + k \cdot T_p$ and the initial condition $(0, 1.5)$. The red curves corresponds to $\lambda_{\pm,1}$ and the green ones to $\lambda_{\pm,2}$.

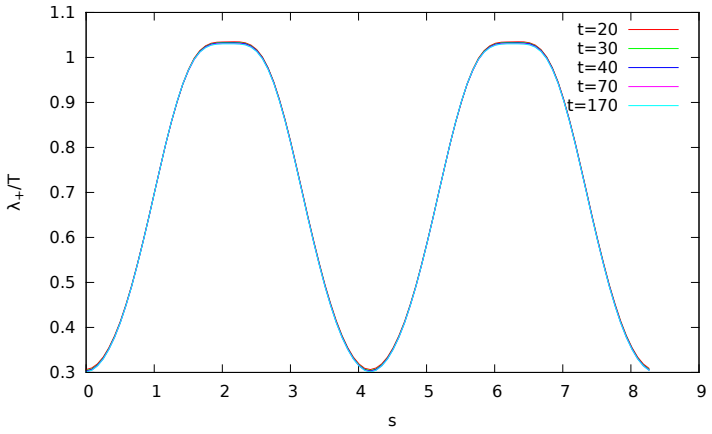


Figure 3.23: Value of λ_+/T as a function of the parameter s along the periodic orbit of the pendulum with initial condition $\vec{x}_0 = (2, 0)$. A point x_s has parameter s if $\varphi(s; t_0, x_0) = x_s$. The value of λ_+ is obtained integrating up to a time T such that it is the closest multiple of the period of the orbit to a fixed time t. Finally λ_+ is divided by the true final time of integration.

initial condition converges to the same value of λ_+/T, but this value varies when the initial point moves, with s, along the orbit. Observe that the points close to the hyperbolic points of the pendulum are the ones which experiment expansions smaller than the ones that are far away, for which the expansion is larger. There are two kinds of expansions: one is a "uniform" expansion along the orbit, which is due to the difference of periods of neighbouring orbits, and there is also a non-uniform and periodic expansion, with the same period as the orbit, which is larger when the velocity of the pendulum is maximal and smaller when the velocity of the pendulum is close to zero. The second expansion dominates the first one, and is the one observed in the plot.

As a final remark, it must be noted that in the libration regime the expansions are bounded, therefore, the value λ_+/T should go to zero as time increases. However, this does not happen because for long integration times T the radius of the initial box r_0 decreases and, as a consequence, since l_+ is bounded, the value $\lambda_+/T = l_+/(r_0\, T)$ is not necessarily going to zero.

3.3.3 Computations in higher dimensions

The above procedures can be extended to higher dimensional systems. In what follows it is shown how to proceed with both the circular and the elliptic R3BP. One of the main difficulties is how to represent the resulting data. As it has already been mentioned, this problem is solved by fixing a value of the energy (E) (or the pseudo-energy ($\hat{E}$)). For the second dimension reduction the hyperplane of section is $y = 0$ instead of $x = 0$. Observe that this change does not affect the qualitative results that we obtain.

In the computation of the contraction and expansion rates, the additional dimensions also affect the way on how the directions of maximum expansion and contraction are determined. There are two possible ways to proceed: the simplest approach is to compute the expansions and contractions only in the directions of the two-dimensional grid used to represent the results; a more accurate procedure would require the exploration of all possible directions in the full phase space. For the first approach, the expansion and contraction directions are computed taking into account neighbourhoods only in the 2D grid directions, i.e. taking a circle in $(x, \dot{x})$, parametrised by the angle θ and fixing $y = 0$ and $\dot{y} = \dot{y}(x, \dot{x}, E_0)$. This is, the perturbation $\vec{\xi}$ to be considered into the polynomial is:

$$\vec{\xi} = (r_0 \cos\theta, 0, r_0 \sin\theta, \dot{y}(r_0 \cos\theta, r_0 \sin\theta, E_0)).$$

Using the angle θ as parameter, one can apply the procedure in a similar way as in the 2D case.

For the second approach, the search must be carried out in the full 4D space. Now the computation of the first approximations of the expanding and contracting directions is done using a parametrisation of the 3-sphere given by three angles $(\theta_1, \theta_2, \theta_3)$. Once suitable initial seeds are obtained, they are refined using a gradient method in order to get more accurate maximum expansion and contraction direction rates. Figure 3.24 shows the expansion and contraction rates in the elliptic R3BP using both approaches. In both cases, one can distinguish those zones of high hyperbolicity for its high expansion and contraction rates.

It has been found that the contraction rates are not as good indicators as the expansion rates, essentially for some numerical issues. The main one is related to the slow convergence rate of the gradient method when there are several close angles for which their associated values are also very close to the minimum. In these cases, the method quickly reaches this set of values but it gets stuck for some iterations. These conflictive cases, where the gradient works with problems, usually happen near the invariant structures to be detected.

In order to speed up the procedure for these ill conditioned cases, some improvements have been implemented. Taking into account that in these cases the iterations of the gradient method almost repeat the same direction search every two iterates, the results are averaged every two iterations getting in this way a better search direction. Also for these cases is a good practice to combine the gradient method with other one-dimensional minimisation procedures, like Newton's method or a golden section search. Newton's method is applied to determine the zero of the first derivative of the system, the golden section search is a variation of the bisection method where, instead of taking the midpoint between the two that has different signs one takes the value that is in the golden ratio. The golden section method is proven to be faster than a usual bisection method when we are minimising or maximising a function (see [PTVF07] for more details). The performance of these implementations is displayed in Figure 3.25. When the gradient method is followed by a 1-dimensional Newton's method, to determine the minimum in the direction of the gradient, a very slow convergence is obtained, as is shown by the blue curve of the figure. When the directions given by the gradient are averaged, the results are much better, although after some iterations (around 400) the difference between two consecutive angles is very small (in fact below the value fixed as one of the stopping criteria of the algorithm). This is shown by the red and green curves of the figure. Nevertheless, if the stop condition is removed, there is a monotonous decrease of the distance function, as is shown by the purple line.

Observe that the values that we obtained are below the double precision of the computer. For that reason the computations for the contraction rates has been done using quadruple precision.

A genetic algorithm to search the maximum and minimum of the distances has been also implemented and tested without any improvement of the results presented up to here. A genetic algorithm starts initializing a random population of points. Then using a fit function of the problem, in our case the maximum or minimum distance, a fitting rate is given to each point. Those points with higher fitting rates are combined and the ones with low rate discarded. In this way we get a new population that have better individuals.

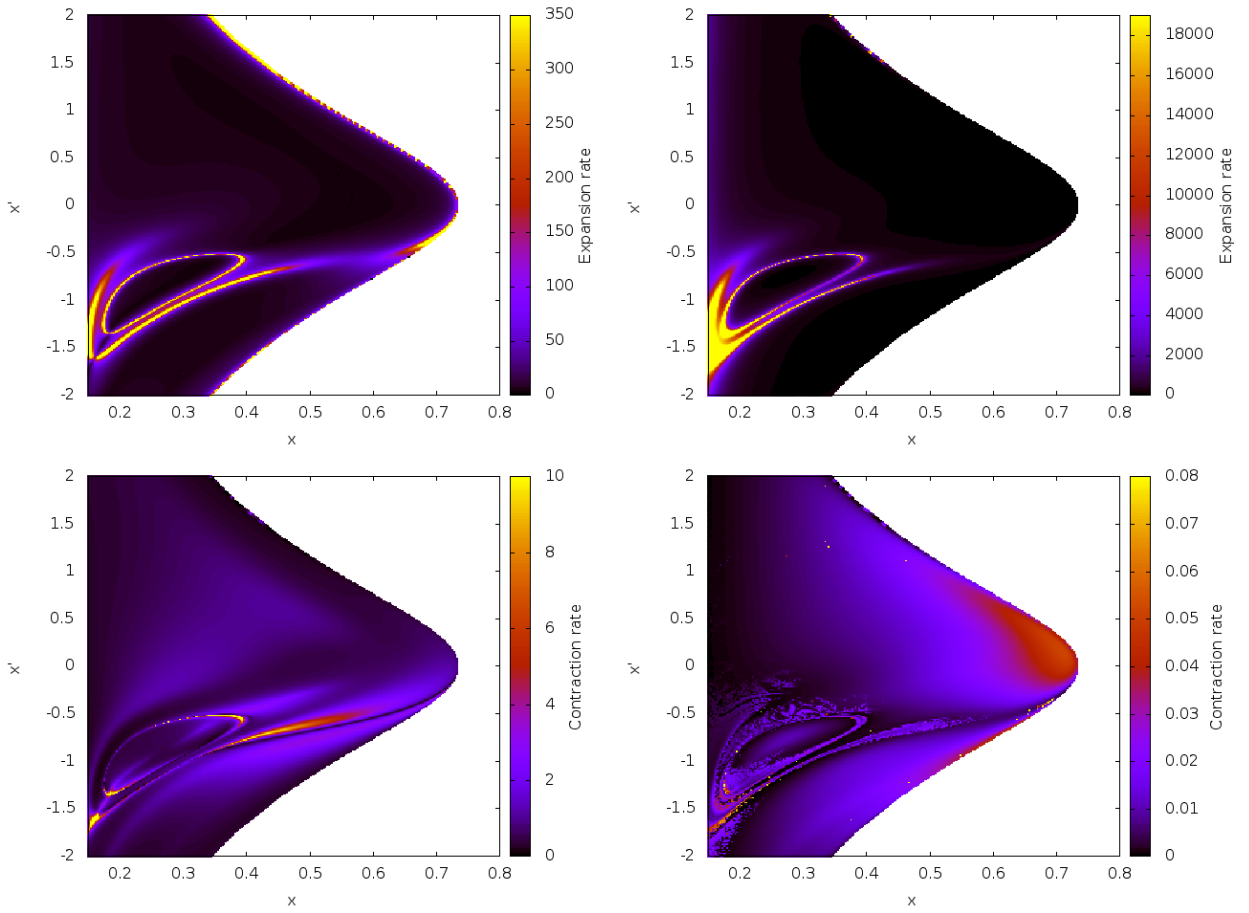


Figure 3.24: Expansion (top) and contraction (bottom) rates for the ER3BP with mass parameter $\mu = 0.1$ and eccentricity $e = 0.04$, taking values in a grid on $(x, y, \dot{x}, \dot{y}) = (x_i, 0, \dot{x}_j, \dot{y}(x, \dot{x}, \hat{E}_0))$, with $\hat{E}_0 = -1.725$. The figures on the left have been obtained using the 2D search, while those on the right doing the search in the full 4D space.

During the procedure it is also possible to introduce new random population and change some values of the individuals to produce variations to the population.

3.3.4 Propagation to the normal space

The dynamical indicator discussed in this section is inspired by the LCS naive idea of "going" to the normal space. The results obtained with this indicator are good, however, there are some situations, that will also be shown, where the indicator fails to give proper results.

The main tool in which this dynamical indicator is based is the computation of the normal repulsion rate: $\rho_{t_0}^{t_0+t}(\vec{x}_0, \vec{n}_0) = \langle \vec{n}_t, D\phi_{t_0}^{t_0+t}(\vec{x}_0)\vec{n}_0 \rangle$. Observe that ρ gives the projection on the normal direction to the material surface $\mathcal{M}$ at time t, $\vec{n}_t$, of the propagation through the flow of a normal vector $\vec{n}_0$ at time $t = 0$, $D\phi_{t_0}^{t_0+t}(\vec{x}_0)\vec{n}_0$. In general, the propagation of $\vec{n}_0$ will not be in the normal space of the final point $\phi_{t_0}^{t_0+t}(\vec{x}_0)$ and, furthermore, its length is, in general, not unitary. According to this, values of ρ grater than one will indicate expansions with respect to the normal direction while smaller than one will indicate contractions with respect to the normal direction.

Recall that the material surfaces are constructed with the orbits of the points of a certain initial $(n-1)$-dimensional surface of initial conditions. Given an initial point $\vec{x}_0$, the normal space, $\mathcal{N}_0$, at $\vec{x}_0$ is spanned

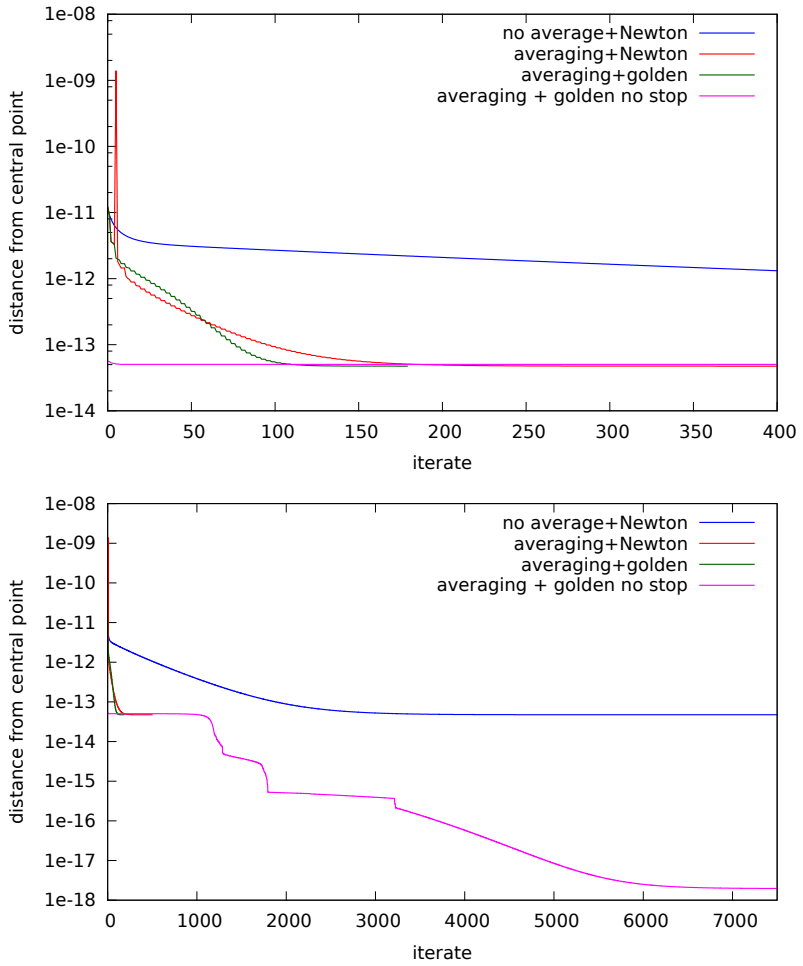


Figure 3.25: Evolution (in quadruple precision) of the minimum distance obtained by the gradient method as a function of the number of iterations for different methodology improvements. Blue: in the standard gradient method a 1-dimensional Newton's method is used to determine the minimum in the direction of the gradient. Red and green: the decreasing direction is computed as the average of the last two gradient iterations and the determination of the minimum along this direction is computed using Newton's method (in red) or a golden section search (in green). The purple line shows the behaviour of the gradient method followed by a golden section search without any stop condition if two consecutive angles are very close. The top plot shows the distance from the centre for the first 400 iterates while the bottom one corresponds to the 7500 first iterates.

(in general) by the $n-1$ normal vectors to the orbit at $\vec{x}_0$, $\vec{n}_{0,i}$, that are orthogonal to the tangent vector $\vec{t}_0$. The tangent, $\vec{t}_T$, and normal vectors, $\vec{n}_{T,i}$, at $\vec{x}_T = \phi(T; t_0, \vec{x}_0)$ define the normal space $\mathcal{N}_T$ at $\vec{x}_T$. The objective is to compute the values of $\vec{t}_T$ and $\langle \vec{n}_{T,i}, D\phi \vec{n}_{0,i} \rangle$.

The first thing to do is to compute the polynomial $P_{T,\vec{x}_0}(\vec{\xi})$ that approximates $\phi(T; t_0, \vec{x}_0 + \vec{\xi})$. This polynomial will give the position of the neighbouring points to $\vec{x}_0$ at time T. However, for most of the values of $\vec{\xi}$, $P_{T,\vec{x}_0}(\vec{\xi})$ is not in the normal space $\mathcal{N}_T$, therefore the states given by $P_{T,\vec{x}_0}(\vec{\xi})$ must be propagated until the normal space is reached.

For the propagation, it is convenient to use local coordinates of the normal spaces at the initial and final points. Let $s_{0,i}$ and $s_{T,i}$ be parametrisations at the initial and final spaces, such that a point $\vec{x}$ on them is

represented by $\vec{x} = \vec{x}_\odot + \sum s_{\odot,i}\vec{n}_{\odot,i}$, where $\odot$ is 0 or T at the initial and final position, respectively.

There are several ways to do the propagation of the states given by $P_{T,\vec{x}_0}(\vec{\xi})$. One possibility is the following: since $\vec{x}_T = P_{T,\vec{x}_0}(0)$ and $P_{T,\vec{x}_0}(\vec{\xi})$ are close, and the tangent direction given by the vector field is a good approximation of the orbit followed by $P_{T,\vec{x}_0}(\vec{\xi})$ to reach the final normal space, one can find the propagated states solving the linear system of n equations with n unknowns:

$$P_{T,\vec{x}_0}(\vec{\xi}) + s_{T,0}\vec{t}_T = \vec{x}_T + \sum_{i=1}^{n-1} s_{T,i}\,\vec{n}_{T,i}.$$

The values of $s_{T,0}$ and $\overline{s}_{T,n}(s_0) = (s_{T,1}, \ldots, s_{T,n-1})$ will be called the tangential distance and the tangential projection, respectively.

Another possibility to compute the propagation is to first determine the rotation that takes the tangent space to the canonical base of the space around $\vec{x}_T$; then, the only thing that remains to be done is to multiply the local basis at $P_{T,\vec{x}_0}(\vec{\xi})$ by the rotation matrix. The results obtained in this way agree with the ones resulting from the other procedure.

The third option is to compute the true distance to the normal space, $\mathcal{N}_T$, and the time of landing, this is: the time required to reach $\mathcal{N}_T$. These computations require the determination of the time expansion of the flow map, which is given by a polynomial $\tilde{P}_{T,x_0}(\tau,\vec{\xi})$ that approximates the flow map $\phi(T+\tau; t_0, \vec{x}_0+\vec{\xi})$, as explained in Chapter 2.

This can be done either computing the polynomials $\tau(\vec{\xi})$ and $\vec{s}(\vec{\xi})$, that give the landing time and the position on the normal space $\mathcal{N}_T$ as a function of the initial deviation $\vec{\xi}$ (using the algorithms given in section 2.4) or solving the system:

$$\vec{x}_t + \sum_{i=1}^{n-1} s_{T,i}\,\vec{n}_{T,i} = \tilde{P}_{T,x_0}(\tau,\vec{\xi}),$$

for each $\vec{\xi}$ in the initial state. Note that now, due to the polynomial term $\tilde{P}_{T,x_0}(\tau,\vec{\xi})$, this system of equations is non-linear and must be solved, for instance, using Newton's method for the $s_{T,i}$ and τ variables. Of course, both procedures are equivalent, and the choice of one or the other depends on the number of iterations required. For a small number of points the second approach is faster, however, if if is required to compute the values of the landing time and the position on the normal space for hundreds of points, then the first procedure is much better.

Observe that the landing time (the time needed to reach the normal space) is closely related with the tangential distance. Since the velocity of all the neighbouring points of x_T is similar, longer distances will require longer landing times. On the other side, since the tangential direction gives a first order approximation of the flow, the distance in the normal space $s(\vec{\xi})$ will give approximately the same result than the tangential projection $\overline{s}_n(s_0)$.

Pendulum computations

Figure 3.26 shows, for the pendulum, the results obtained for the functions $s_{T,1}$ and τ using different initial conditions. From the figure it follows that the approximations obtained using the above methods are good in a relatively large range of initial angular velocities $y(s_0)$ of the pendulum. However, when the value of $y(s_0)$ is close to the one associated to the separatrix ($y(s_0) = 2$), the approximation breaks down. This is due to the fact that the polynomial $P_{T,\vec{x}_0}(\vec{\xi})$ does not provide a good approximation of the flow when de points where it is evaluated are at both sides of the separatrix. Another fact that is worth to be mentioned is that, although only the values of $y(s_0)$ approaching $y(s_0) = 2$ are influenced by the separatrix, the polynomial approximation also breaks down when $y(s_0)$ moves in the opposite direction. This is frequent in this kind of approximations and indicates that, in general, one cannot determine which breakdown is due to the presence of the separatrix .

Figure 3.27 shows the intervals of convergence around the initial conditions $\vec{x}_0$ along the normal to $\vec{x}_0$. Observe that the closer we are to the invariant manifold, the smaller the *convergence zone* to the connected component containing the initial point is.

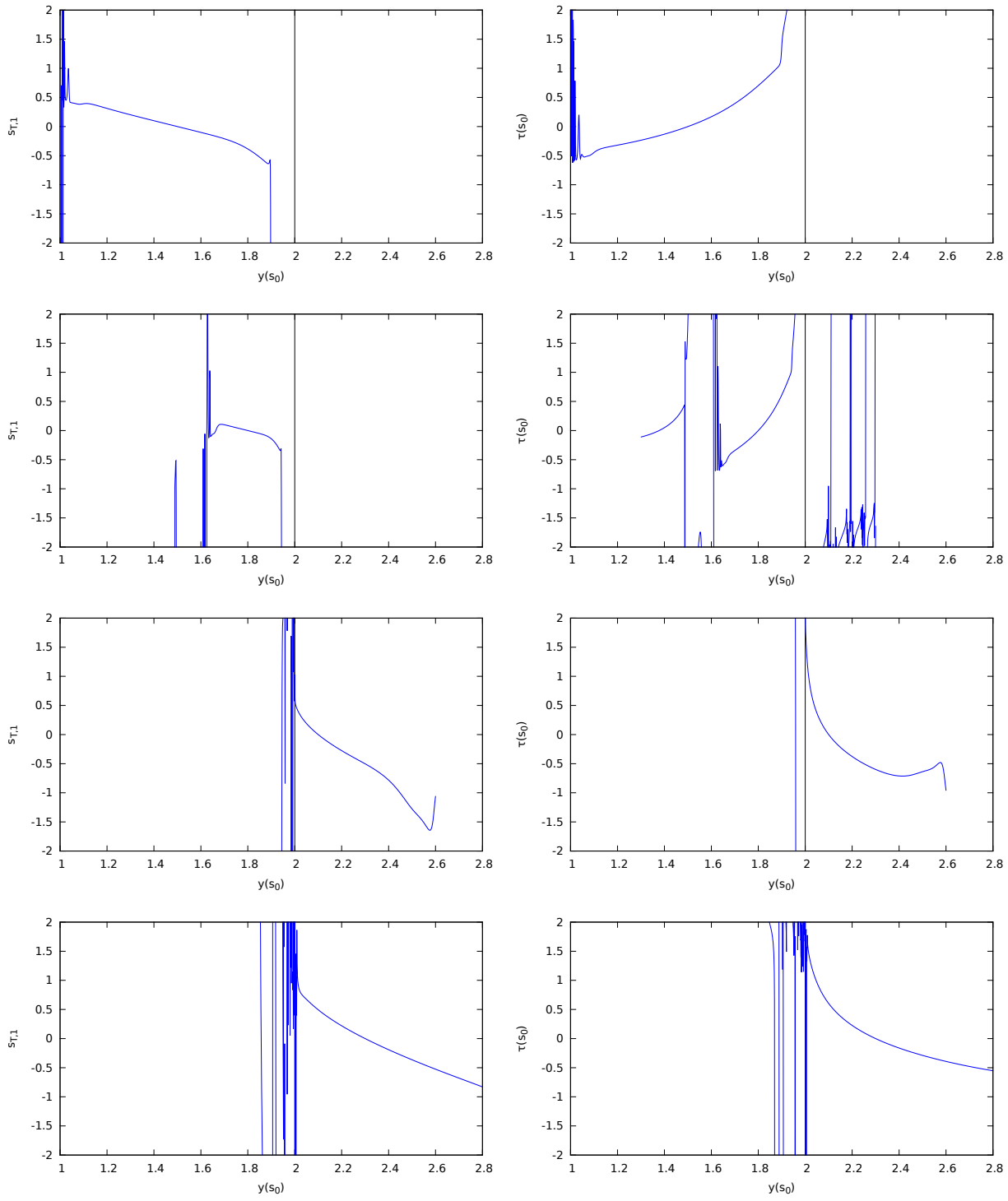


Figure 3.26: The left hand-side plots show for the pendulum the behaviour of the function $s_{T,1}$, after half a revolution, corresponding to initial points of the form $(x_0, y_0 + s_0) = (0, y(s_0))$ for $(x_0, y_0) = (0, 1.5)$, $(0, 1.8)$, $(0, 2.1)$ and $(0, 2.3)$ (from top to bottom). In all the plots, the initial angular velocity is represented on the x-axis. The plots on the right hand-side represent the function τ for the same initial points, parametrisation and final times. The separatrix is always at $y(s_0) = 2$.

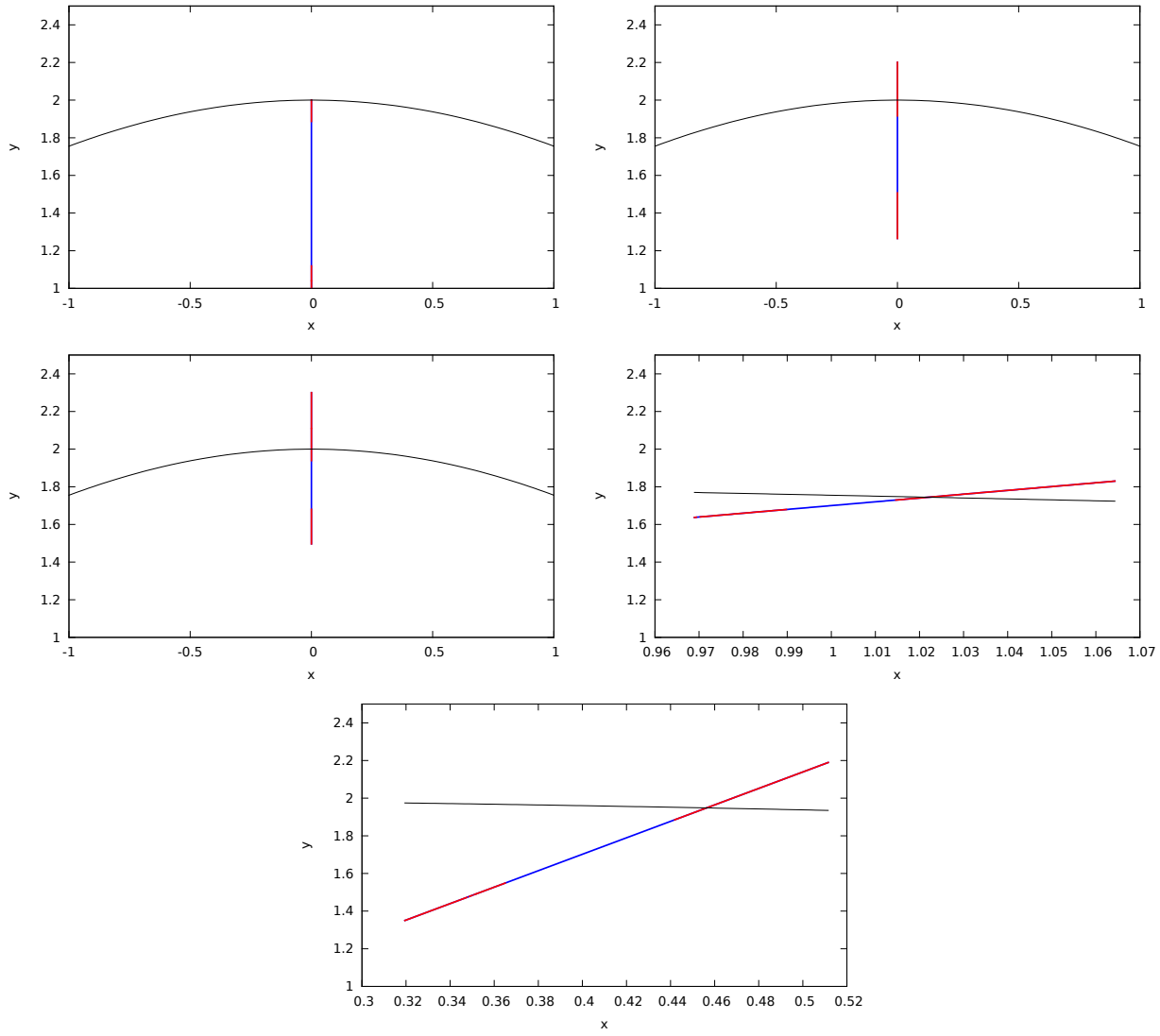


Figure 3.27: From left to right and top to bottom, convergence intervals, along the normal direction, associated to the initial conditions of the unperturbed pendulum: $\vec{x}_0 = (0, 1.5), (0, 1.7), (0, 1.8), (0.4, 1.7), (1, 1.7)$. The black curve is the separatrix, the red points are those which do not converge, and the blue ones those with convergence. All the initial conditions are integrated during half a period (which is a long enough time-interval to get close to the hyperbolic point).

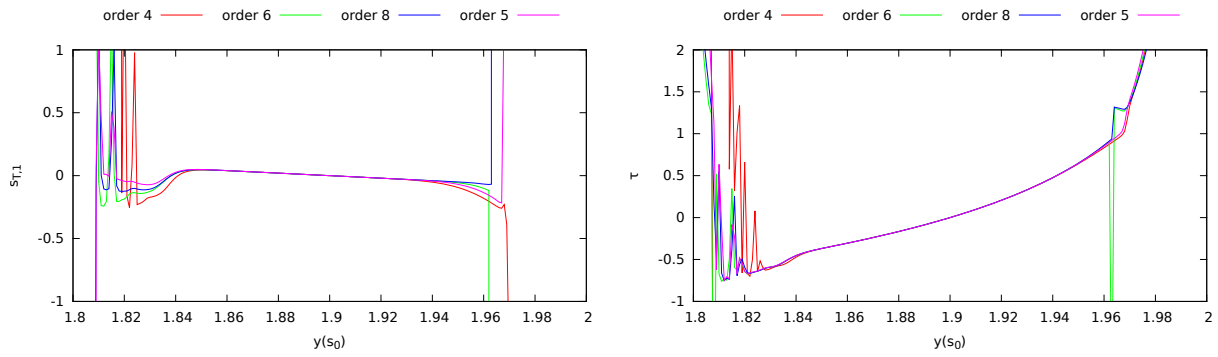


Figure 3.28: Behaviour of the functions $s_{T,1}$ (left) and τ (right) as a function of $y(s_0)$ for different maximum orders of P_{T,x_0}. In all the computations the initial condition is $\vec{x}_0 = ((0, 1.9)$ and the integration time interval T is half a revolution.

In order to increase the convergence zones one can use higher order terms in the polynomial P_{T,x_0}. This is what is shown in Figure 3.28 where the behaviour of the functions $s_{T,1}$ and τ as a function of $y(s_0)$ is shown for different maximum orders of P_{T,x_0}. It can be seen that when the order increases the convergence zone enlarges. In all the cases divergences start abruptly.

For the perturbed pendulum there appear some difficulties. The first one has to do with the way in which the normal space to the orbit is defined. At a first glance, one can select it using two options:

- defining the normal direction as the direction perpendicular to the flow in the two-dimensional phase-space (x, y),

- the same as above, but in the 3-dimensional extended phase-space (x, y, t).

Since the second option has some representation problems, most of the results that follow correspond to the first one.

Using the normal direction as the one perpendicular to the flow in the 2D space, the plots of Figures 3.29 and 3.30 show the behaviour of the propagation of initial conditions in $\mathcal{N}_0$ up to the normal space $\mathcal{N}_T$ at a number of different epochs T =0.2, 0.8, 1.2, 1.3, 1.6 and 1.8, the right plots are magnifications of the left ones. The small blue segment around $\vec{x}_0 = (1, 1)$ is the set of initial conditions in $\mathcal{N}_0$. The purple segment is the image of this segment under $P_{T,x_0}(\vec{\xi})$. The propagation of the same initial conditions up to $\mathcal{N}_T$, computed using Newton's method, is represented by the black segment. The green and red curves are the stable and unstable manifolds, respectively, of the fixed point $(\pi, 0)$. In the sequence of plots shown in the figure, one can see that there are some values of T (1.2 and 1.6) for which the propagation up to the normal space $\mathcal{N}_T$ does not converge to something reasonable. This happens when there is some quick turn in the orbit (grey curve) Observe that in this last case there is no nearby invariant manifold that can explain why this breaking is produced.

It must be noted that the existence of values of T for which Newton's method diverges is mainly due to the way in which the approximation to the Normal space is done. There is always a set of points in the neighbourhood of the initial point $\vec{x}_0$ for which there is convergence. In some cases, when the neighbourhood enlarges there exist a value $\vec{\xi}_{ta}$ for which there is a tangency between the flow of $\vec{x}_0 + \vec{\xi}_{ta}$ and the normal space $\mathcal{N}_T$. The flow map of the values grater than $\vec{\xi}_{ta}$ does not intersect $\mathcal{N}_t$ and, then, the Newton method does not converge or it does to wrong values. Figure 3.31 shows a graphical explanation of the reason of non-convergence for the presented procedure.

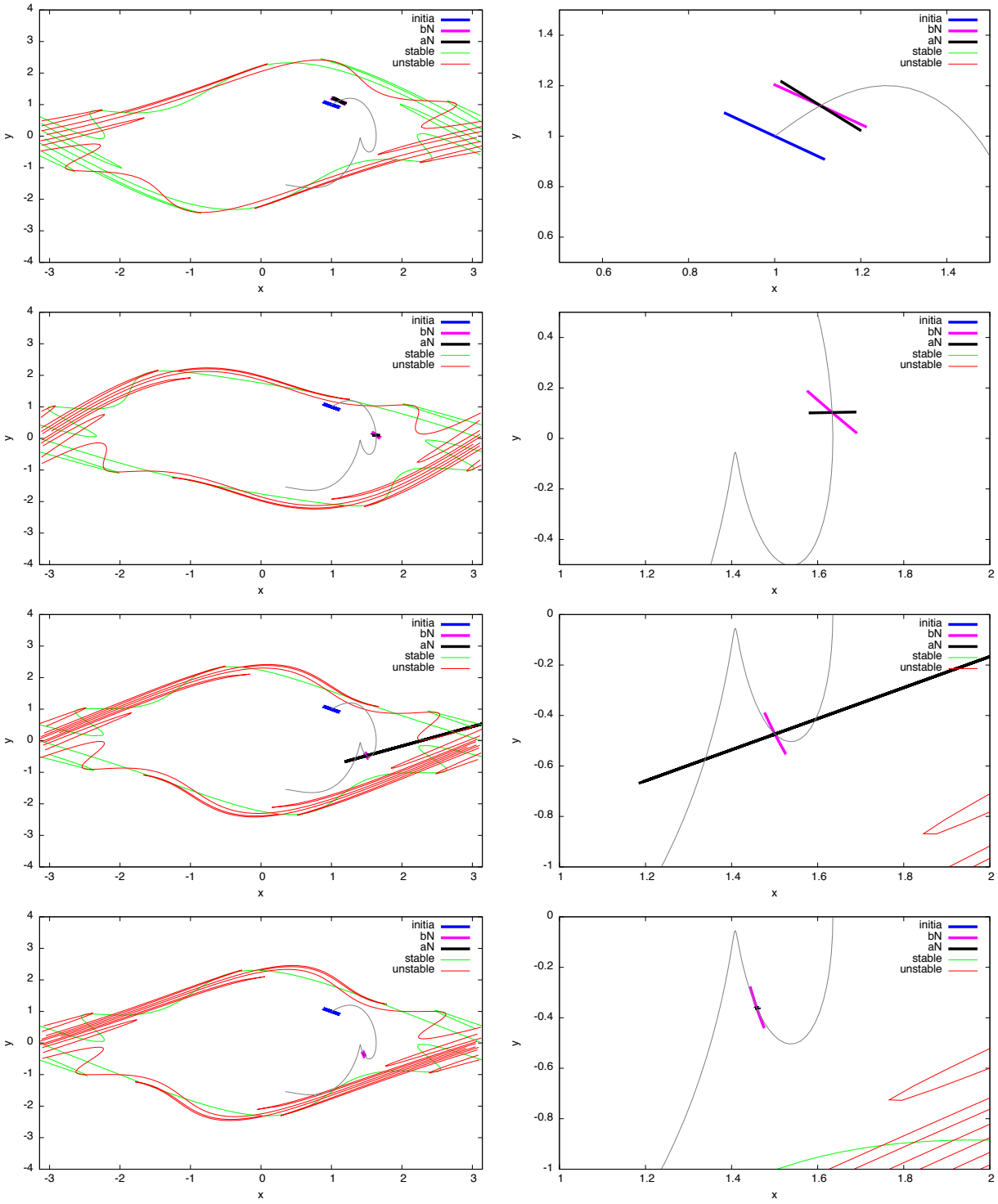


Figure 3.29: From top to bottom, propagation of the normal space at $\vec{x}_0 = (1, 1)$ up to $T = 0.2$, 0.8, 1.2, and 1.3. The plots at the right part are magnifications of the plots at the left. See explanations in the text.

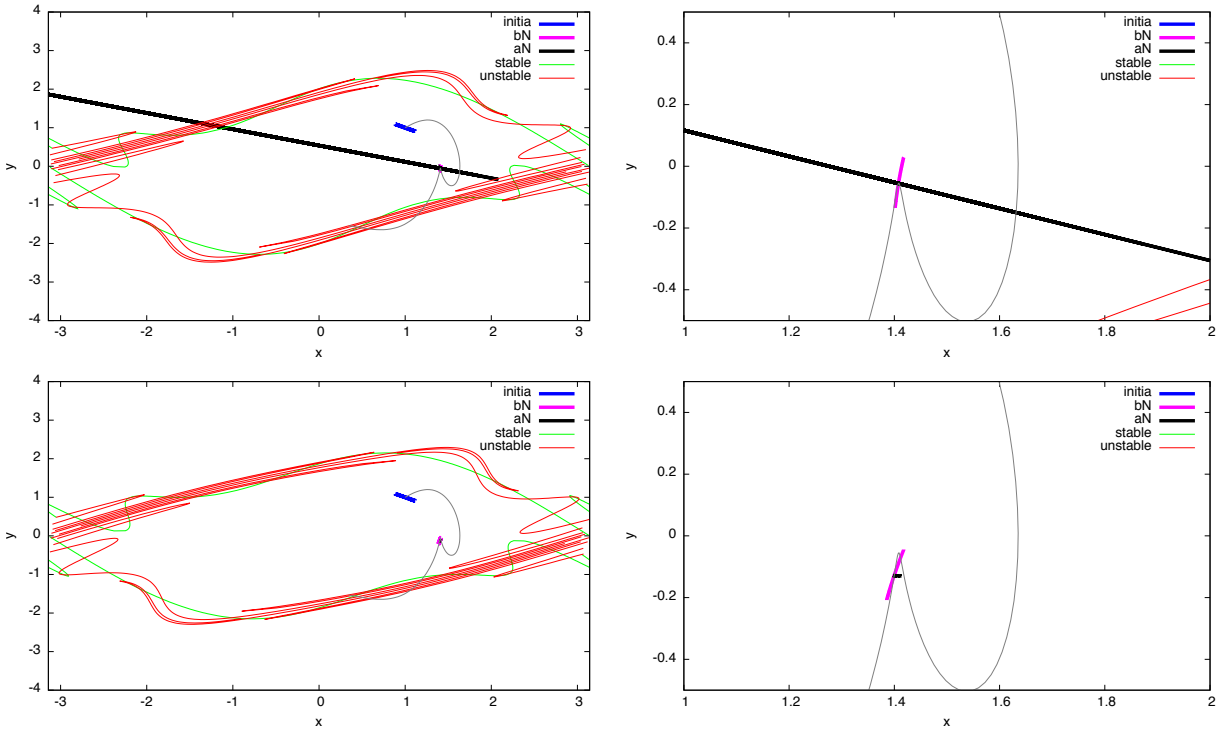


Figure 3.30: Continuation of Figure 3.29, from top to bottom, propagation of the normal space at $\vec{x}_0 = (1, 1)$ up to $T = 1.6$ and 1.8. The plots at the right hand are magnifications of the plots at the left. See explanations in the text.

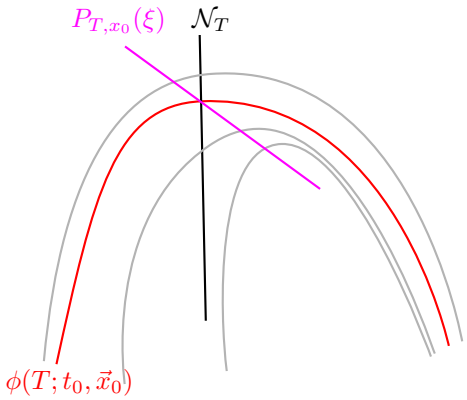


Figure 3.31: Graphical representation of the non-convergence zones observed in Figures 3.29 and 3.30. The red curve is the flow map of $\vec{x}_0$. Around it, there are represented the flow maps of three neighbouring points of the form $\vec{x}_0 + \vec{\xi}$. The innermost one does not intersects the normal space $\mathcal{N}_T$, because of that, the Newton procedure does not converge.

RTBP computations

As it happens with the other indicators, in the RTBP there is an additional difficulty due to the dimension four of the model. Using Jacobi's first integral, the dimension of the normal spaces $\mathcal{N}_0$ and $\mathcal{N}_T$ can be reduced in one unit proceeding as follows:

- Fix an energy level in $\mathcal{N}_0$. This will reduce in one unit the dimension of both $\mathcal{N}_0$ and $\mathcal{N}_T$, since all the propagated points are in the same energy level.
- In the arrival normal space $\mathcal{N}_T$, compute the distances from the propagated points to $\vec{x}_T$ and, in order to see if there is a contraction or expansion, compare these distances with those between the initial points and $\vec{x}_0$.

The propagations up to the normal space are done as follows:

1. Compute the normal space, $\mathcal{N}_0$, to the orbit departing from $\vec{x}_0$ using a Gram-Schmidt procedure for the determination of $\vec{n}_1$, $\vec{n}_2$ and $\vec{n}_3$ such that:
$$\vec{x} \in \mathcal{N}_0 \implies \vec{x} = \alpha\vec{n}_1 + \beta\vec{n}_2 + \gamma\vec{n}_3.$$
2. Select those points in $\mathcal{N}_0$ that are in the same energy level, $E(\vec{x}_0)$, as $\vec{x}_0$. This can be done in the following way:
   - First, fix values of α and β.
   - Using Newton's method, solve the equation $E(\vec{x}(\alpha,\beta,\gamma)) - E(\vec{x}_0) = 0$ to get $\gamma = \gamma(\alpha,\beta)$.
3. Using the jet, propagate up to a certain time T the initial condition $\vec{x}(\alpha,\beta,\gamma)$. As in the case of the pendulum, the propagation up to $\mathcal{N}_T$ of the points in the neighbourhood of $\vec{x}(\alpha,\beta,\gamma)$ can be done either using Newton's method or computing the polynomial that gives the image points in $\mathcal{N}_T$.
4. Compute the ratio between the final distance of the propagated points to $\vec{x}_T$ and the distance of the initial points $\vec{x}(\alpha,\beta,\gamma)$ to $\vec{x}_0$. This ratio will be denoted by $r(\alpha,\beta)$.

The function $r(\alpha,\beta)$ plays the role of the function $s_{T,1}(y(s_0)$ introduced for the pendulum, with the only difference that it depends on two variables, α and β, instead of only one.

The function $\gamma(\alpha,\beta)$, solution of the equation $E(\vec{x}(\alpha,\beta,\gamma)) - E(\vec{x}_0) = 0$, must be such that $\gamma(0,0) = 0$. So, for its numerical computation, it is convenient to start at $(0,0)$ and to take points in the (α,β) plane spiralling outwards from the origin. For each new point, and in order to start Newton's method, one can use the previously computed value of γ. This prevents the possibility of computing a slice of the constant energy surface for which the $\gamma(0,0) = 0$ condition does not fulfil.

Figure 3.32 shows examples of the normal expansion function $r(\alpha,\beta)$. In the same figure, the invariant manifolds of the Lyapunov periodic orbits around L_1 with $E = -1.583043$ are also displayed (in green the stable manifold and in red the unstable one). Observe that if the integration is done forward in time, then the black zones confine the stable manifold while if its done backwards in time then the black zones reveal the unstable manifold. This fact has sense since:

- For integrations forward in time, $T > 0$, the stable direction is where the orbits spend more time close to the periodic orbit. For very short forward propagations, the orbits will get closer to the periodic orbit (since it is in the stable manifold); then, since the approximation is not perfect, the unstable part will move away incrementing the expansion rate. Therefore dark zones for forward in time integrations should be stable.
- For integrations backwards in time, $T < 0$, the unstable direction is where the orbits spend more time close to the periodic orbit. For very short backwards integrations, the orbits will get closer to the periodic orbit (since now we are moving around the unstable manifold). Then, the stable part will move the integration away from the periodic orbit and the expansion rate will increase, therefore dark zones for backwards integrations should be unstable.

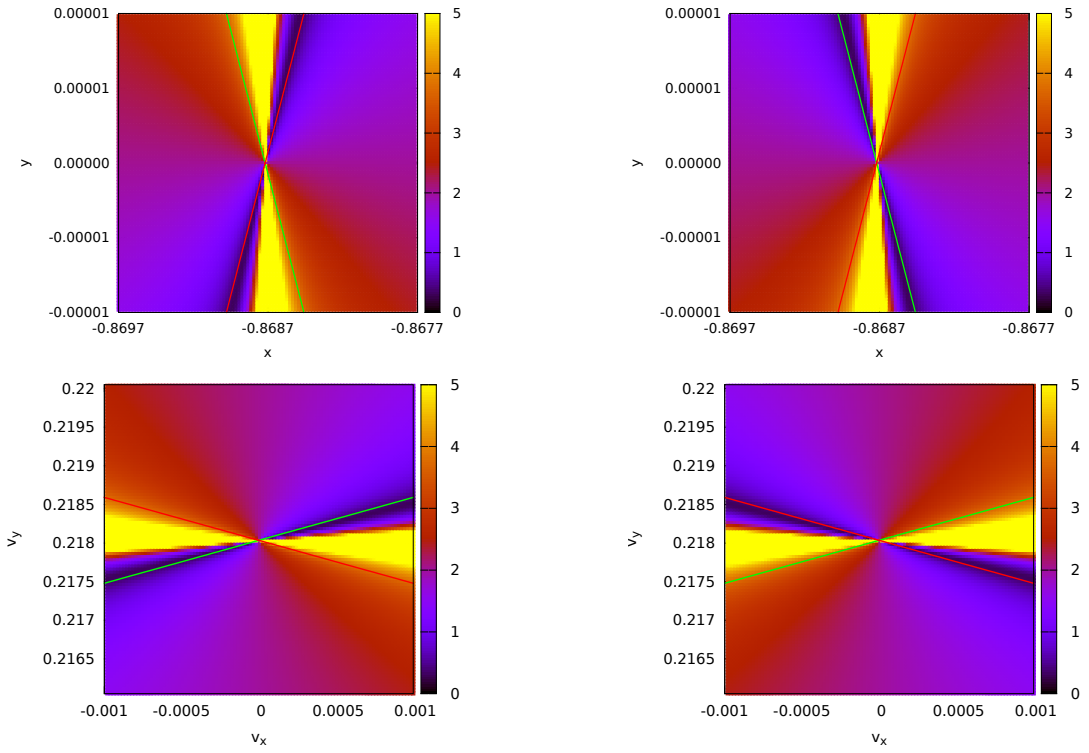


Figure 3.32: Behaviour of the normal expansion function r in the vicinity of the normal space $\mathcal{N}_0$ for forward and backward integrations. The two top figures display the behaviour of the r as a function of the physical coordinates x, y, and the bottom ones as a function of the components of the velocity: v_x, v_y. The central point, $\vec{x}_0$, is the initial condition of the Lyapunov periodic orbit around L_1 with $E = -1.583043$. The final integration times are $T = \frac{-T_p}{10} \approx -0.283228$ (left) and $T = \frac{T_p}{10} \approx 0.283228$ (right), where T_p denotes the period of the orbit. The green and red lines are the projections of the intersection of the invariant manifolds with $\mathcal{N}_0$.

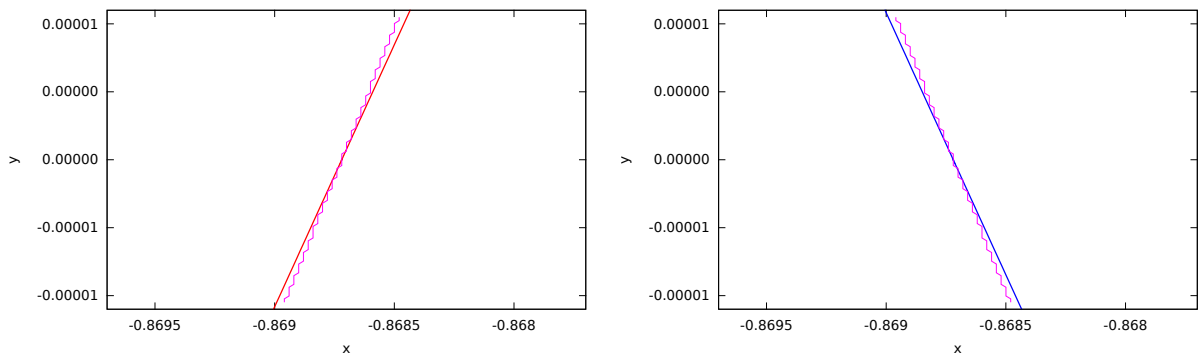


Figure 3.33: For each column of the grid used in Figure 3.32 for the computation of the function r, y-coordinate of the point of the column where the value of the function has a minimum. The left plot corresponds to backward integrations and the right one to forward integrations. The continuous segments are the projections of the intersection of the invariant manifolds with $\mathcal{N}_0$.

For each column of the grid used in Figure 3.32 for the computation of the normal expansion r, Figure 3.33 shows the point at which the value of the function is minimum. One can see that the agreement between lines of minima and the stable and unstable manifolds, also displayed in the figure, is very good.

Figure 3.34 shows the behaviour of the function $r(\alpha, \beta)$ when the (x, y) coordinates of the point $\vec{x}_0$ of the Lyapunov periodic orbit is fixed (and equal to the one used in the previous computations) and different values for the argument of the velocity are used. Of course, the modulus of the velocity is fixed in order to fulfil, in all the cases, the energy condition. There, one can see the difference between the second (top right plot) and the other ones. In this plot, the high values of r (that correspond to the expanding zones) are in a thin region, while in the other plots the high value regions are in wider ones. The sharp transitions observed in this plot are due to the fact that the initial conditions considered are in a regime where there is a change of the dynamics. The white empty region of the bottom left image is a zone where there it do not exists values of $\gamma(\alpha, \beta)$ verifying the energy condition.

Figure 3.35 shows the logarithm of relation between the maximum value and the minimum value of the normal expansion r as a function of the argument of the velocity for the initial configuration state $(x, y) \approx (-0.8687, 0)$ for different final times, $T = T_p/10$, $T_p/4$. and T_p, where T_p is the period of the Lyapunov periodic orbit that we are considering. With the appropriate initial velocity that point belongs to the Lyapunov periodic orbit considered in the previous plots. There, one can see that the high values of r are related to changes of the dynamics. Figures 3.36 and 3.37 shows the orbits of some consecutive initial velocities (right) related to the coloured segment of the left plot. Each colour corresponds to a different orbit. The high values of the relation between the maximum and the minimum normal expansions of the left plots are related with changes of the shape of the orbit. For instance, the first high values differentiates those orbits that goes to the Earth's region of those orbits that goes to the Moon's region. In between, there is a small decreasing given by the orbits that stays close to the periodic orbit.

In Figure 3.35 it is also possible to appreciate that the as the time increases, the maximum values of the relation plotted are bigger. Also appear more spikes of maximum values. Both effects have sense. When the time increases there are more possible behaviours for the orbits, and, therefore, there will be more changes between them which explains the aparition of more spikes. On the other side, those orbits that were behabing in a different way, when the time increases, will be more different, enlarging the relation of both indicators.

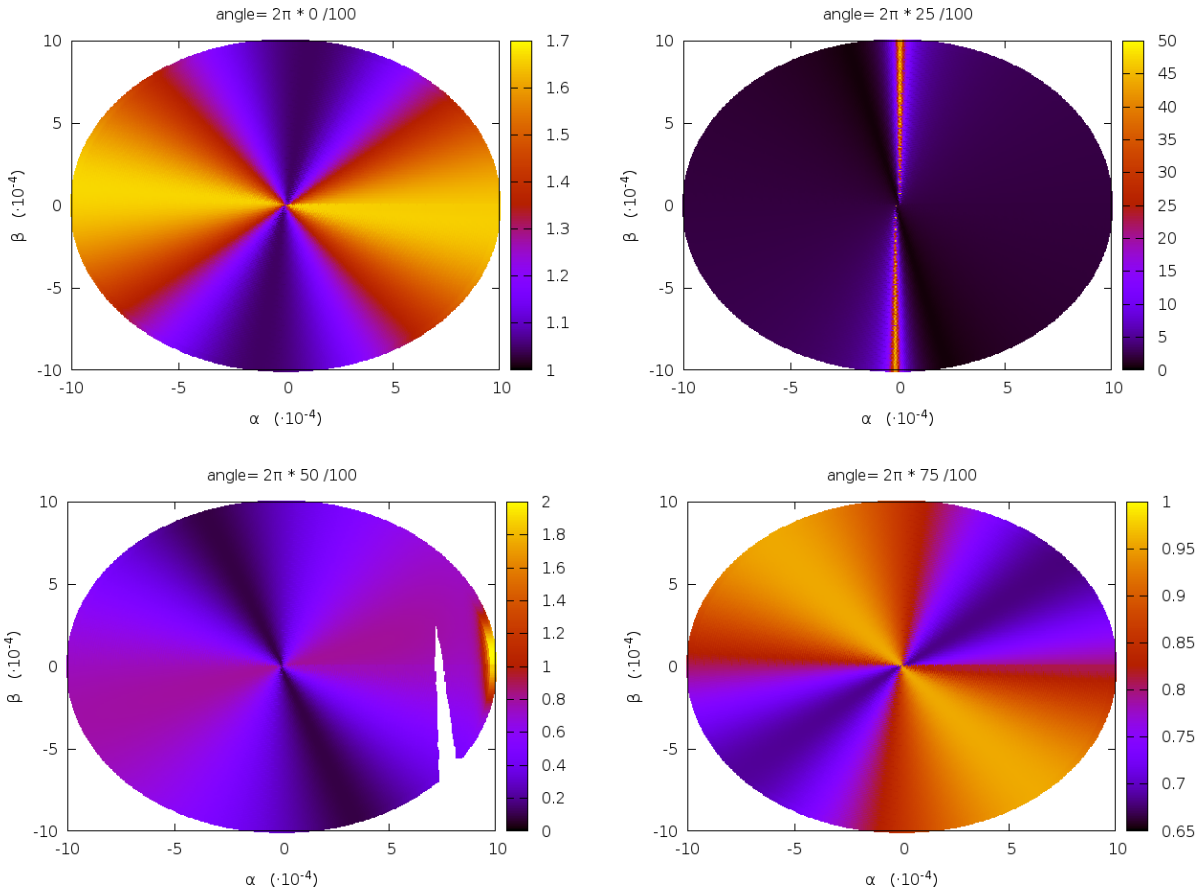


Figure 3.34: Behaviour of the normal expansion $r(\alpha, \beta)$ when the (x, y) coordinates of the point of the Lyapunov periodic orbit are fixed (and equal to the one used in the previous computations) and different values for the argument of the velocity (0, $\pi/2$, π and $3\pi/2$) are used.

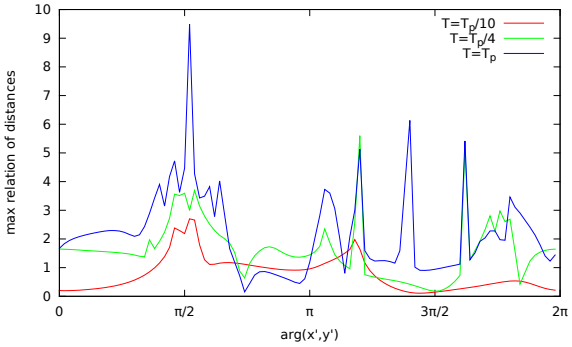


Figure 3.35: For the Lyapunov periodic orbit around L_1 with energy $E = -1.583043$, maximum value of the maximum expansion and the minimum expansion in the normal space $\mathcal{N}_0$ as a function of the argument of the initial velocities at the initial configuration space $(x, y) \approx (-0.8687, 0)$. The final time of computation is $T = \frac{T_p}{10} \approx 0.283228$ (red), $T = \frac{T_p}{4} \approx 1.132912$ (green) and $T = T_p \approx 2.83228$ (blue). The angle of the initial is represented on the x-axis.

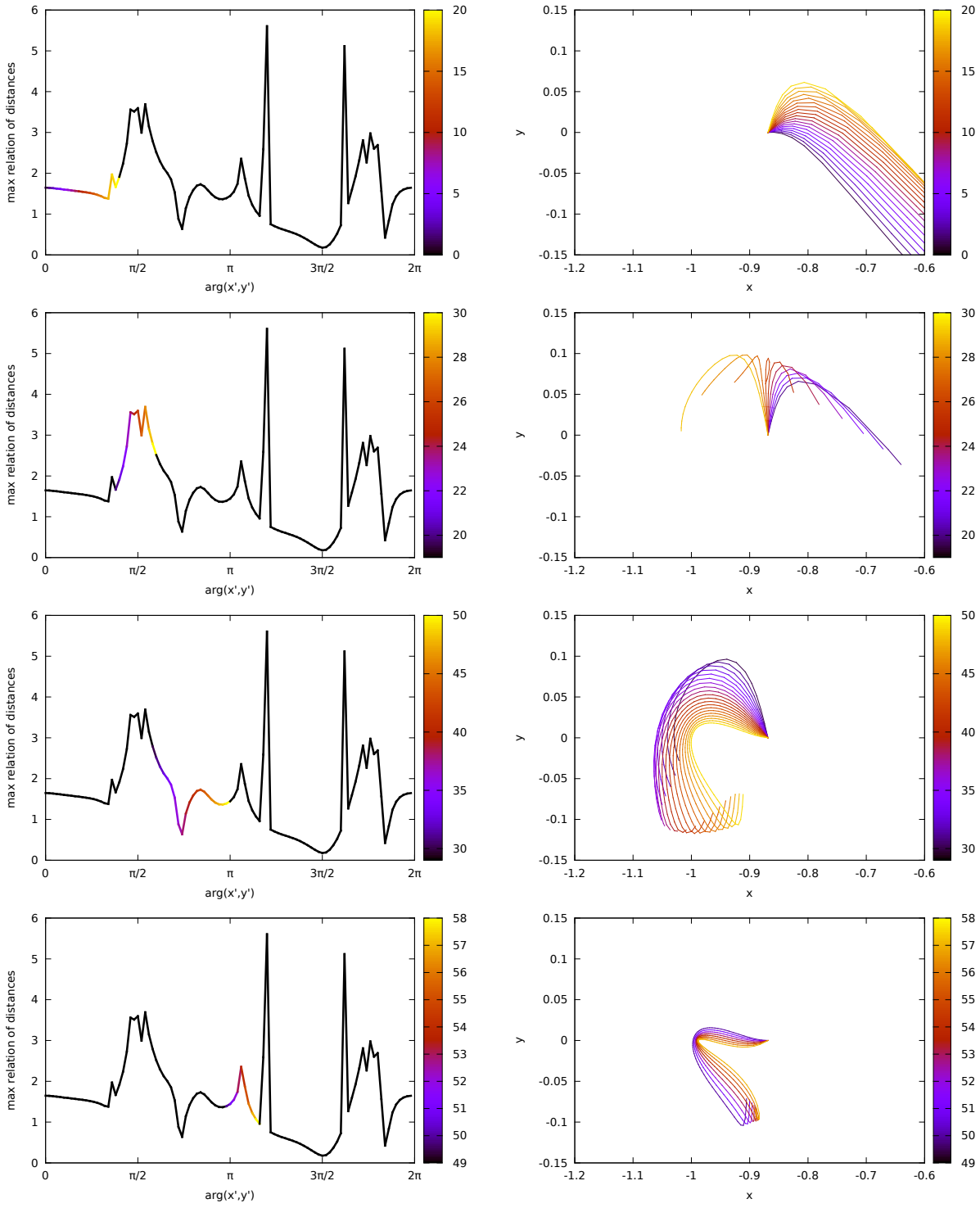


Figure 3.36: Left: For the Lyapunov periodic orbit around L_1 with energy $E = -1.583043$, maximum value of the maximum expansion and the minimum expansion in the normal space $\mathcal{N}_0$ as a function of the argument of the initial velocities at the initial configuration space $(x, y) \approx (-0.8687, 0)$. The final time of computation is $T = \frac{T_p}{4} \approx 1.132912$. The coloured segment is related to the righ plots. Right: orbit of initial conditions related with the colour given in the left plot. Each colour represents an initial argument of the velocity, the configuration space initial condition is the one used in the right plots.

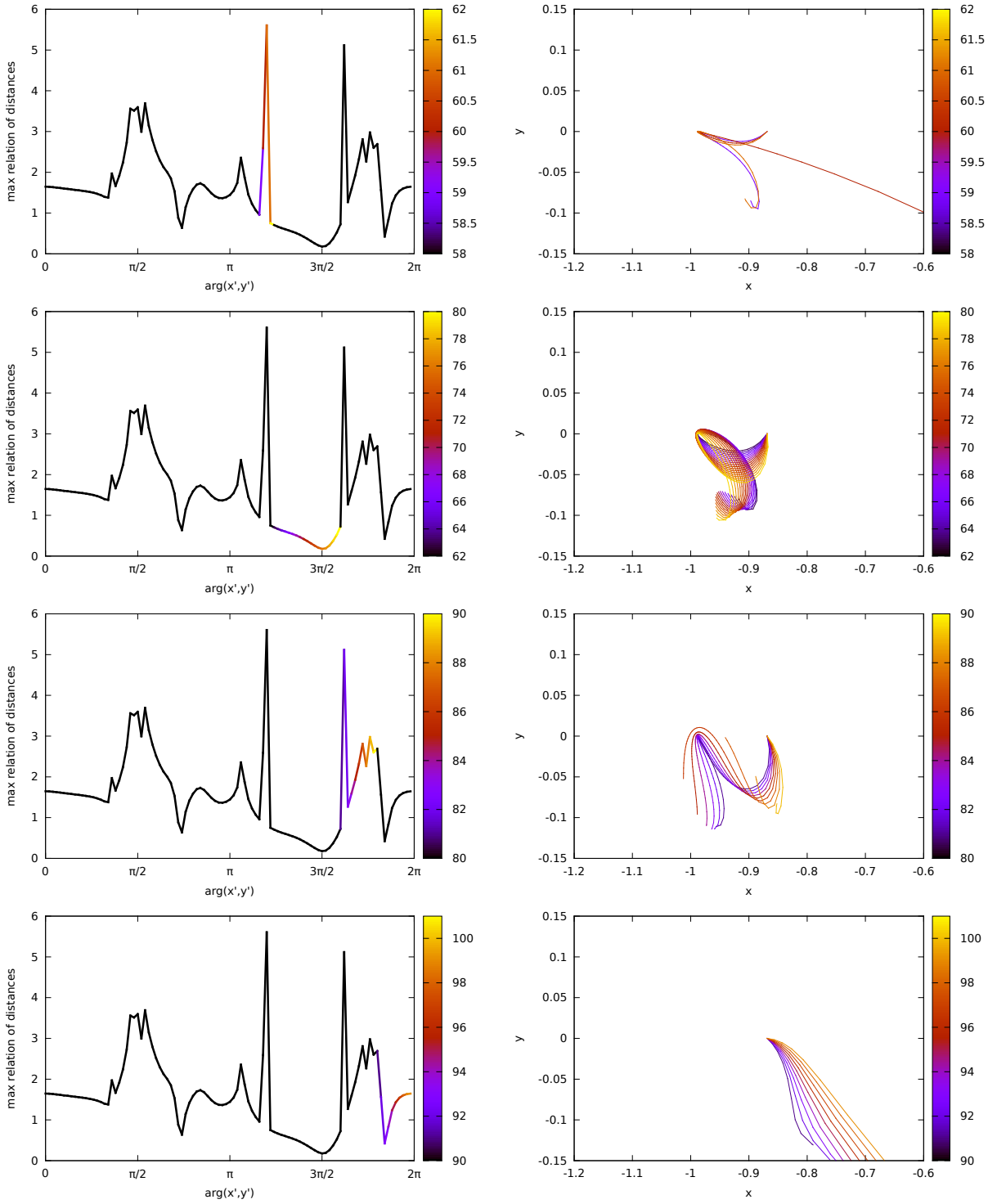


Figure 3.37: Continuation of Figure 3.36

3.4 An structure extraction algorithm

In order to see how the LCS evolve when changing one of the parameters of a model, for instance in initial true anomaly in the ERTBP, a visualisation algorithm has been developed that allows the extraction of the desired structures from different data plots.

Algorithm 4 (Extraction of structures from functions defined on 2-dimensional grids)**.**

1. Let λ be the function defined on the grid under consideration.
2. Compute the function λ for all the points of the grid. The finer the grid the better the results will be.
3. Fix a tolerance Δ and discard all the points such that $\lambda(x) < \Delta$. This is the condition to be applied in the case that we are looking for the structure defined by the maximum values of λ. For other other definitions of the structure, the algorithm can be easily modified.
4. Give the same index, i_{cc}, to all those points that are in contact, this is, at the same connected component of the grid. In this way, several sets of connected points are obtained.
5. For each point x_0 at each connected component, then:
   (a) Assign to x_0 an index i_γ that will identify the "curve" γ (that can have a certain width) to which x_0 belongs.
   (b) Look within the 8 neighbouring points of x_0, which is the one that has the highest value of λ. Define it as x_1.
   (c) Select x_{n+1} as the point with the highest value of λ, within the 3 neighbouring points of x_n opposed to x_{n-1}. In Figure 3.38 (top) the yellow square represents x_n, the blue square x_{n-1}, and the three green ones those where the point x_{n+1} must be searched (the red points are not tracked).
   (d) Continue the procedure until we arrive to an edge of the grid or the path reaches a previously selected point x_s. In the first case, the index i_γ is not modified and is assigned to all the points of the sequence. If the second situation happens, the index associated to all the points of the sequence will be the one already assigned to x_s.
6. Shrink the width of the "curves" according to the following procedure:
   - Assign to each point x on a curve γ the value N of neighbours (including itself) that x has.
   - If $N = 1$ discard x and proceed with the selection procedure at the next point.
   - If $N = 2$ look to the neighbour of x, if for this neighbour N is again equal to 2, discard both points, otherwise keep x and proceed with the next point.
   - If $N = 3$ it is possible to continue the path through the 3 points(it is the ideal situation). Proceed with the next point.
   - If $N = 4$ or $N = 5$ there are possible bifurcations. To solve them and look at the values of N at neighbour points. If at any of them $N = 3$ then keep the point, otherwise discard it.
   - If $N > 5$, then there are to many points to follow a path. Look at the values of λ at the 8 neighbouring points, discard the $N - 4$ with the smaller values of λ, and continue the procedure as in the previous case. Figure 3.38 (bottom) shows an example of the discarding procedure of this step.

   After the above selection procedure, all the points have 3,4 or, at most, 5 neighbours.

7. Two neighbour points, x and y, are said to be *directly connected* if they are in the same row, column or diagonal. The points x and y are *connected* if one can go from x to y through directly connected points. After this classification, all the points will be on a curve γ of connected points. In this step, assign to all the points of γ a new index i_γ.
8. Select the connected curves that we are interested to extract (user's decision). These curves may have bifurcations.
9. Select the bifurcations that we are interested in (user's decision). This will generate a curve close to be closed.
10. Compute the baricenter of all the points selected.
11. Parametrise the selected points by the angle between a fixed direction and the segment joining the baricenter with the point.
12. In order to have a smooth representation of the results, use interpolation, with respect to the angle, to add additional points to the grid of the ones selected.

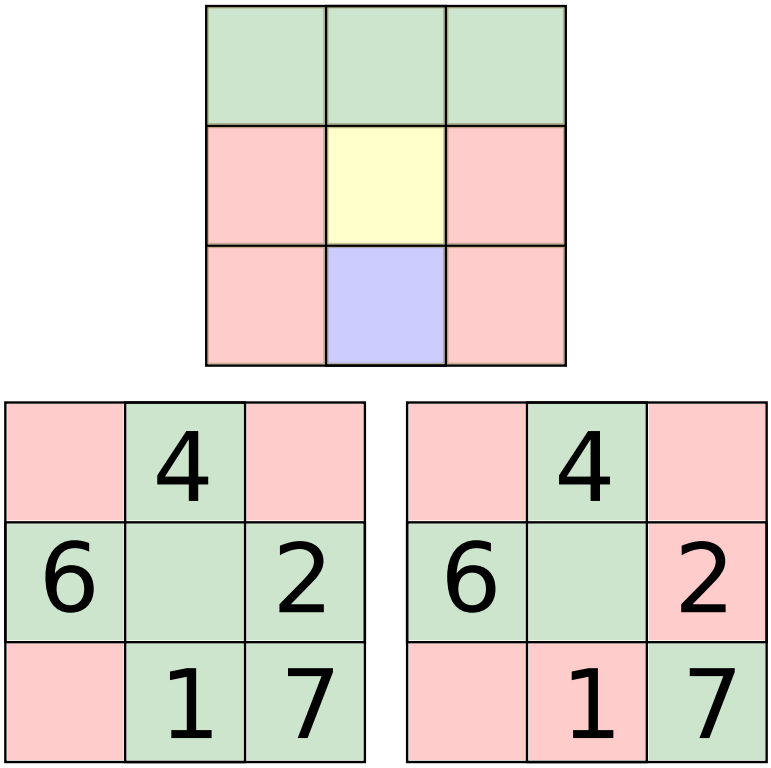
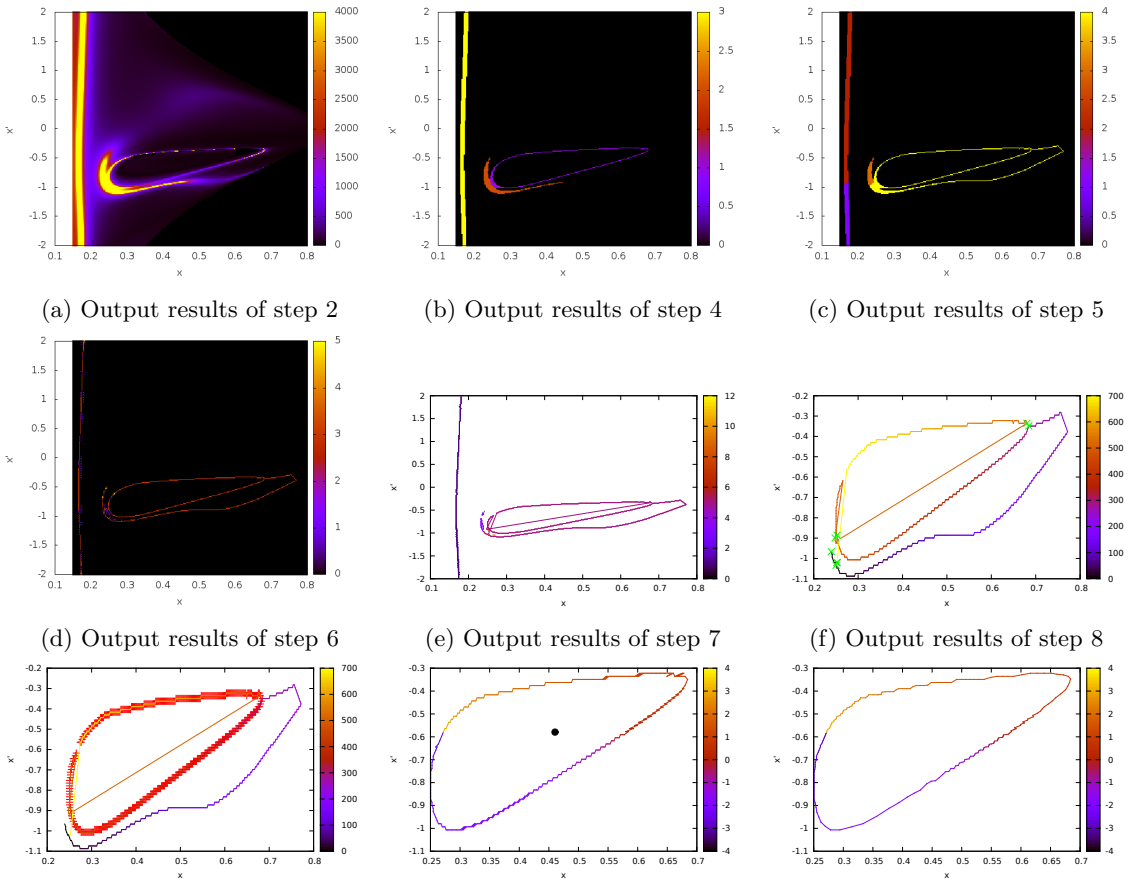


Figure 3.38: Top: example of step 5 of the extraction algorithm. The yellow square represents the actual point x_n, the blue one the previous point x_{n-1}, and the three green squares are the candidates to be x_{n+1} where the function λ will be evaluated. The red squares are not tracked. Bottom: Example of the shrinking step 6i with $N = 6$. The left plot shows the central square associated to x and the remaining 5 squares with the associated values of λ. The right hand plot shows the squares that have been discarded according to the shrinking procedure.

Figure 3.39 shows the results of several steps of the extraction procedure for the ERTBP with $e = 0.04$, for a fixed value of the initial true anomaly ($f_0 = 0$). The expansion rate has been selected dynamical indicator and the procedure looks for maximum values of it. $\tilde{E}_0 = -1.6$.



(a) Output results of step 2 (b) Output results of step 4 (c) Output results of step 5

(d) Output results of step 6 (e) Output results of step 7 (f) Output results of step 8

(g) Output results of step 9 (h) Output results of step 10 (i) Final result of the procedure12

Figure 3.39: Example of the steps of the extraction algorithm of the structures obtained from the pseudo-energy level $\hat{E}_0 = -1.6$ for the ERTBP with eccentricity $e = 0.04$, mass parameter $\mu = 0.1$ and initial phase $f_0 = 0$.

Figure 3.40 shows the final results for the ER3BP with a fixed value of eccentricity $e = 0.04$, mass parameter $\mu = 0.1$ and two different values of the initial phase: $f_0 = 0$ and $f_0 = \pi/2$. The initial pseudo-energy $\hat{E}_0$ is represented in the z-axis. These results are in good agreement with the ones obtained in [GMDTC09] using a completely different procedure.

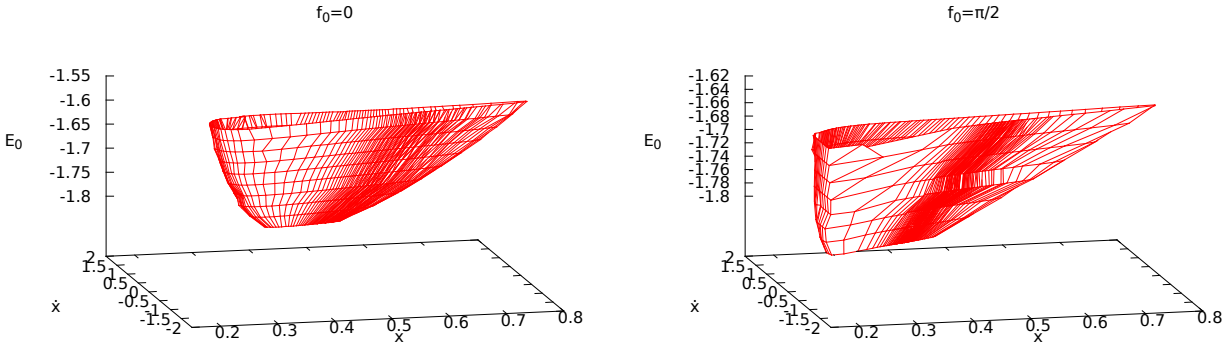


Figure 3.40: Intersection with the plane $y = 0$ of the LCS corresponding to the time-dependent analogue of the stable manifold of a periodic orbit around the L_1 point in the ER3BP. The initial true anomalies are $f_0 = 0$ (left) and $f_0 = \pi/2$ (right). In both plots the eccentricity is $e = 0.04$ and the mass parameter $\mu = 0.1$. The structures have been computed using the maximum expansion rate as indicator. The initial conditions x, $\dot{x}$ have been taken inside a grid with $y = 0$ and $\dot{y} = \dot{y}(\hat{E}_0)$. The range of pseudo-energies used moves from $\hat{E}_0 = -1.8$ to $\hat{E}_0 = -1.58$ in the first case and from $\hat{E}_0 = -1.8$ to $\hat{E}_0 = -1.64$ in the second one.

3.5 Detection of practical stability regions

It is well-known that the triangular equilibrium points of the CR3BP are linearly stable for values of the mass parameter μ below the Routh's critical value μ_1. It is also known that in the spatial case they are nonlinearly stable, not for all the initial conditions in a neighbourhood of the equilibrium points L_4, L_5 but for a set of relatively big measure. The fraction of stable motions tends to 1 when the size of the neighbourhood tends to 0, for almost all $\mu \in (0, \mu_1)$. This follows from the celebrated Kolmogorov–Arnold–Moser theorem. In fact, there are neighbourhoods of computable size for which one obtains "practical stability", in the sense that the massless particle remains close to the equilibrium point for very long time intervals.

An important question is which part of this stability subsists when the idealised CR3BP is substituted by the Earth–Moon system with its real motion and under the very strong influence of the Sun and the milder perturbations due to planets, solar radiation pressure, no spherical shape of the Earth and Moon, etc. This has been studied since long time ago in [MS81, GJMS01, SSST13]. In [SSST13] it is shown how the invariant manifolds of L_3 enclose the practical stability zones.

These zones of practical stability can be detected using the FTLE as well as any other of the indicators presented in this paper. Here are shown the results obtained when one considers initial conditions in a grid of points in the $x - y$ plane with zero initial velocities ($\dot{x} = \dot{y} = 0$).

The results for the maximum expansions, using the 4D search, are shown in Figure 3.41 for positive $T = 30$ (left) and negative $T = -30$ (right) final times of integration. To avoid extremely long integration time intervals, initial conditions close to the main primaries have been avoided, as well as those far from them, which correspond to stable orbits. In the figures, one can detect the stable zones as those associated to low expansion rates. Note that these zones are not symmetric with respect to the x-axis. This asymmetry can be explained because the indicator depends on the integration time interval as well as if the integrations has been done forward or backwards in time. In both figures, one can see two large regions around L_4 and L_5, located at $(1/2 - \mu, \pm\sqrt{3}/2 = \pm 0.866..)$, corresponding to the practical stability regions described in the references already mentioned, as well as other stable zones produced by other mechanisms.

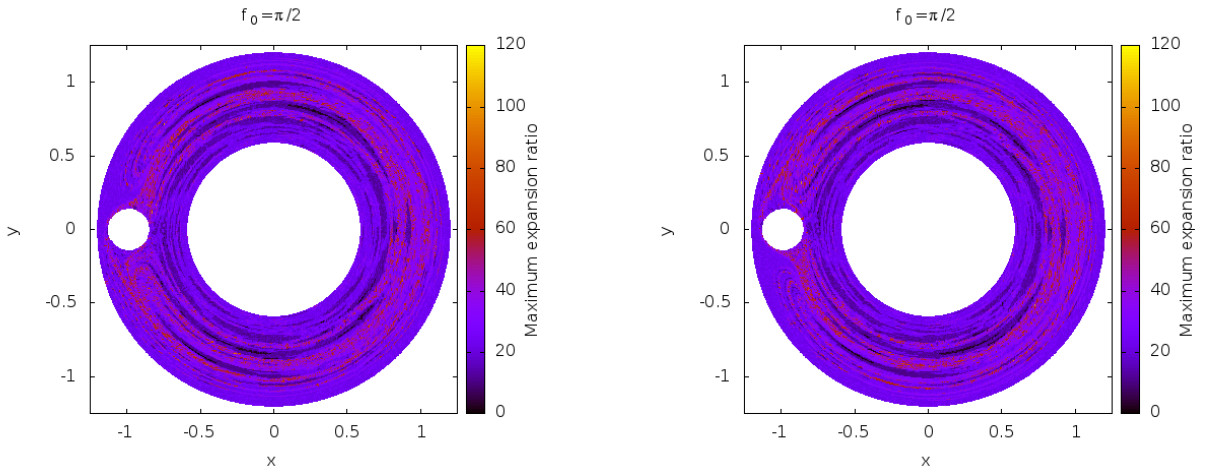


Figure 3.41: Maximum expansion for the CR3BP in forward time (left) and backward time (right). There appear stable zones (darker regions) around the big primary and close to $L_{4/5}$ points.

Forward time integrations are useful to reveal stable hyperbolic structures while backwards ones to reveal unstable ones. Both hyperbolic structures can be seen in Figure 3.41.

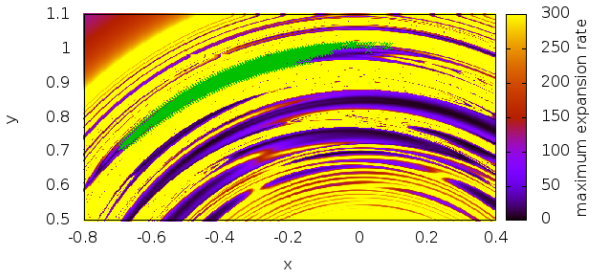


Figure 3.42: Zoom around the L_5 point showing the practical stability region using the maximum expansion rate as indicator. The stability region computed in [GJMS01] is also displayed (in green).

Figure 3.42 shows the a magnification of the stable region around L_5. The practical stability region computed in [GJMS01], using a completely different approach, is also displayed (in green) to show the good agreement between these two completely different procedures. The same plot shows other stable regions, some of them are even bigger than the stability zone around L_5, and are analysed with more detail in what follows.

Figure 3.43 shows the average and the standard deviation of the semi-major axis of all the orbits represented in Figure 3.44 considered as Keplerian orbits with respect to the larger primary. There one can see how, indeed, the orbits with smaller standard deviations are those ones related to the stable zones while the ones with big deviations are close to the unstable zones. The big deviations are produced when the perturbation of the small mass affects the orbit in a notorious way and, therefore, is far from a Keplerian motion. Therefore, the stable zones are filled with orbits that are never close to the small primary.

In Figure 3.44 the dark green region corresponds to the already known practical stability zone around L_4. The motions in the red region are close to almost periodic orbits (precessing Keplerian ellipse) around the large primary, with a basic period close to three times the period of the small primary (2π). Analogously, the motions in the light-green region are close to a almost periodic orbits, with a basic period close to $5 \times 2\pi$. The ones in the cyan region are again precessing ellipses with a basic period close to $2 \times 2\pi$. When we approach

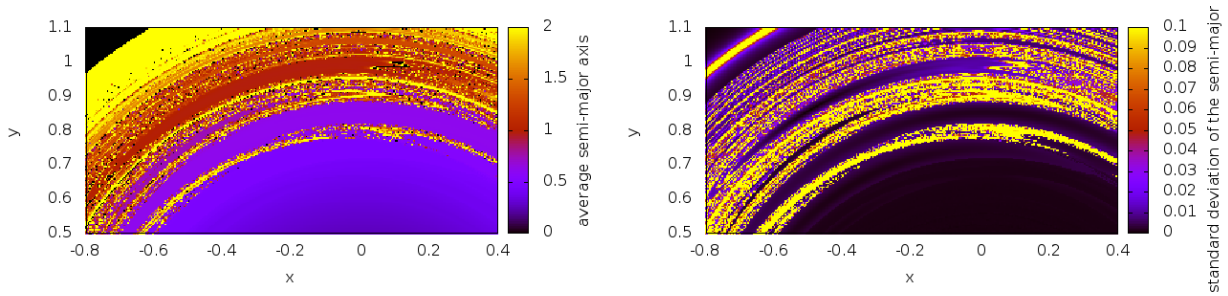


Figure 3.43: Average (left) and standard deviation (right) of the semi-major axis of the orbits around the practical stability region considered as Keplerian orbits with respect to the larger primary.

the unstable region the motion described becomes more irregular, as can be seen in the orbits displayed in Figure 3.44. The ones in the boundaries are not as close to be periodic as the ones in the centre of the stable region. The unstable zone between both regions is filled with orbits that due to the gravitational influence of the small primary, are far from a precessing Keplerian ellipse.

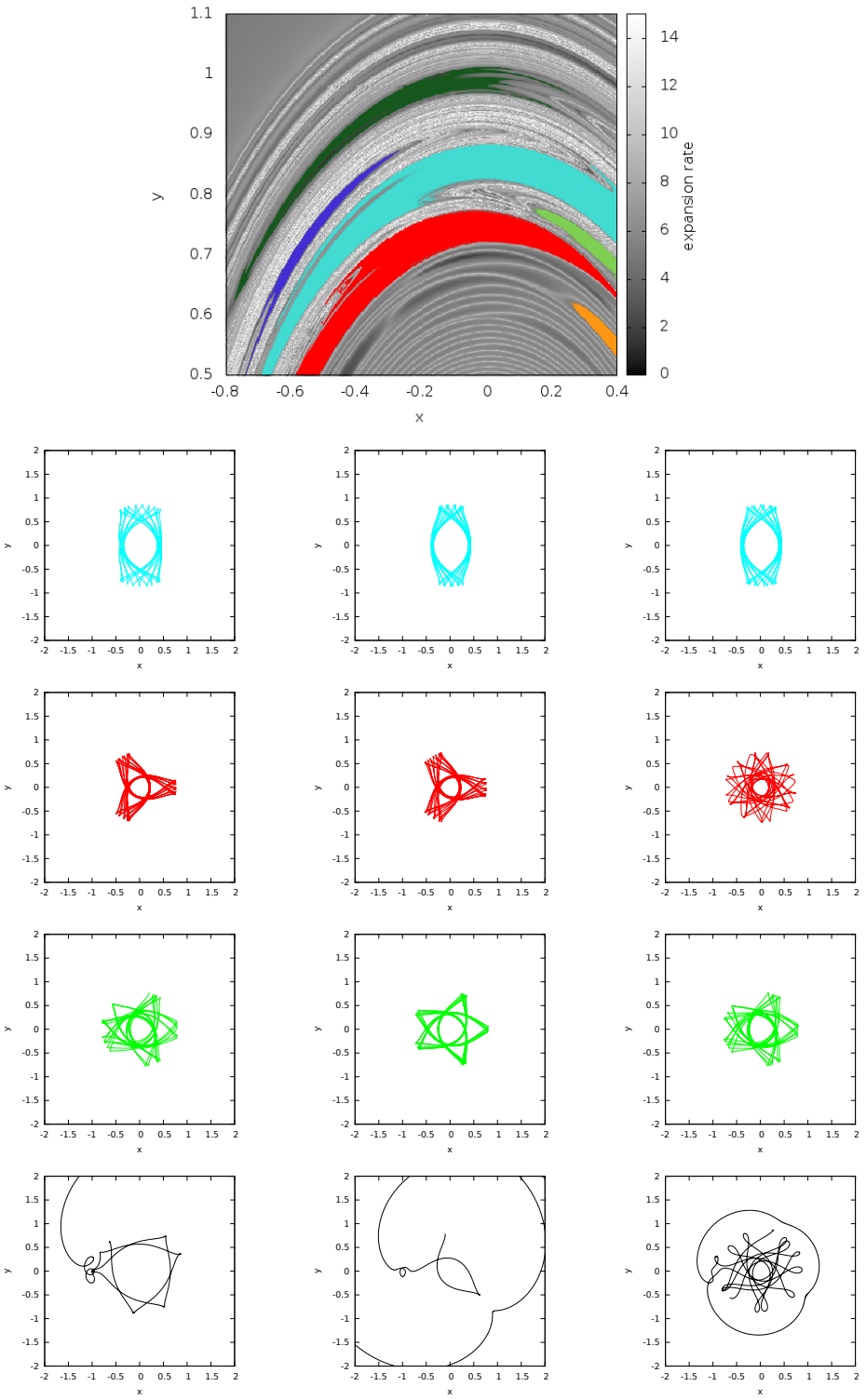


Figure 3.44: Top: Maximum expansion rates around L_4 computed with a final time $T = 30$. The coloured highlighted regions correspond to stability regions, the dark green on the top is the practical stability region around L_4. The coloured orbits displayed have initial conditions in the region of the same colour. The last row corresponds to orbits in the unstable regions.

Chapter 4

Satellite collision probability

In this chapter we will deal with the motion of an spacecraft around the Earth. As is well known, this motion is mainly ruled by the Earth gravity field that, at first order, it can be considered as the central force field of a point mass, but there are other perturbations that need to be taken into account. When the spacecraft is close to the Earth's surface the oblateness and other gravitational perturbations as well as non gravitational ones, like the atmospheric drag, affect the orbits. For spacecrafts in higher altitudes, the gravitational effect of the Sun and Moon should also be taken into account. The solar radiation is also important for high orbits with satellites almost always exposed to the Sun, but also for low orbits with frequent shadows.

For a given physical model and suitable initial conditions for a spacecraft, we can propagate the orbit by means of numerical integration method. However the initial state, and maybe some parameters in the model, will not be accurately determined. Usual methods to deal with these uncertainties fall in two different groups: on one side, the state transition matrix is used to obtain fast linear approximations. On the other side, Monte Carlo schemes are used to evaluate higher order nonlinearities. Using methods in the first group the results can be inaccurate, while using the ones in the second ones can be too slow. In this chapter we present the Jet Transport as a third method combining both needs: the evaluation of high order perturbations and the speed of the computations.

The equations modelling the motion of an spacecraft about the Earth have been widely studied. A common model is to consider the main body, in this case the Earth, as Kepler problem. Then we add perturbations due to the non spherical shape of the Earth by means of the spherical harmonics, $C_{n,m}$, $S_{n,m}$, where n and m are the degree and the order of the spherical harmonics that we are considering. In [MG05, Kau66] these models are introduced and the authors provide tables of coefficients for the Earth gravitational field.

Besides the gravitational effect of the Earth, the motion of the spacecraft is also perturbed by other forces like the gravitational forces of the Sun and Moon, the atmospheric drag or the solar radiation pressure. All these forces that affect the spacecraft can be introduced in the model and their effect considered in the jet computations. In order to consider different models, we label them according to the perturbations that we consider in the Keplerian problem in the form,

NNMM(perturbations)

where *NN* states for the degree and *MM* the order of the spherical harmonics that determine the gravitational field and *"(perturbations)"* is a list of characters denoting other included perturbations:

- *s*: Sun perturbation
- *l*: Moon perturbation
- *d*: Atmospheric drag
- *r*: Solar radiation pressure

Table 4.1 shows some examples of the models implemented and the corresponding name following this notation.

Since we are trying to produce fast an accurate integrations a good system of units is also necessary. A good choice for it is to maintain all the magnitudes in a similar range during the integration. In this way we avoid possible cancellations or terms that do not add information to our equations because they are too small. In the following *canonical units* will be used. These units are selected in such a way that the unit of distance (DU) is the distance of the equatorial radius of the Earth (or the planet that is being considered), $R_\oplus = 6378.140$km. The time unit (TU) is defined in such a way that the speed of an object in a 1 DU circular orbit is 1 DU/TU. With this choice the period of this orbit is $T = 2\pi\ TU$ and from Kepler's third law,

$$\frac{4\pi^2}{T^2} = \frac{GM}{R^3},$$

the gravitational parameter is $GM = 1DU^3/DU^2$.

In the first section of this chapter we introduce in detail different perturbative models that have been considered. Later these models are studied and compared in terms of computational effort and accuracy in order to choose the best options for our study. An special effort has been put in adapting these models to the Jet Transport. Some of them have a direct adaptation, while others require more attention in some details. Tests and checks of the software implementation are also included.

In a second section we present the application of the Jet Transport to the propagation problem under uncertainties. The Iridium-Kosmos collision happened in February 2009 is used as a test bed case for a probabilistic analysis using Monte Carlo procedures. Finally we finish this chapter introducing some open problems as well as some ideas on how to address them using the Jet Transport.

4.1 The model

In this section we introduce the basic model as well as different included perturbations. As it has been stated, our starting basic model is the result of the Newtonian law of gravitation,

$$\vec{F} = -\frac{GMm}{r^3}\vec{r},$$

where $\vec{F}$ is the force on the spacecraft, G is the gravitational constant and M and m are the masses of the massive body (Earth) and the spacecraft respectively, $\vec{r}$ represents the position vector.

Using this law it is well known that the potential associated to the motion of an spacecraft under an spherical homogeneous Earth has potential $U_0 = GM/r$ and corresponds to Keplerian motion. However, deviations from the homogeneous sphere are relevant in the motion, specially for low Earth orbits, and have to be taken into account. Other perturbative forces are the gravitational pull of other massive bodies in the Solar System. In the case of the Earth the more important bodies are the Sun and the Moon, the first one because of its mass, the second one because its proximity.

Finally the last perturbative force introduced in our research is the atmospheric drag that comes from the friction of the spacecraft and the atmosphere. All these perturbations are discussed in the following. In addition we also compare between variants of the same perturbation effect to see which is the more convenient to our interests. For instance, to take into account the third body perturbations due to the Sun,

Model	mn	mm	Drag	Sun	Moon	Solar rad.
0200	2	0				
0200s	2	0		✓		
1010sld	10	10	✓	✓	✓	

Table 4.1: Examples and notation for the different models used.

we have to determine the Sun's position and several options could be available: from a simple circular model to a more precise model coming from the interpolation of an ephemerides file. Being the second option more precise it is much slower, when probably the gain in accuracy is not relevant. Issues like this will be also discussed in this chapter.

4.1.1 Earth gravity field

In order to take into account the effect of having a geoidal planet instead of an spherical one, usually perturbations are expressed by means of spherical harmonics. A general description can be found in in [Kau66, MG05], being the main idea to expand the gravitational potential of the Earth using Legendre polynomials that can be evaluated using recursive formulae and obtaining in this way an accurate vector field. Let us consider the Earth potential expanded in spherical harmonics and given by the formula:

$$U = \frac{GM_\oplus}{R_\oplus} \sum_{n=0}^{\infty} \sum_{m=0}^{n} C_{nm} V_{nm} + S_{nm} W_{nm}. \tag{4.1}$$

Here C_{nm} and S_{nm} are coefficients that have been fitted experimentally, and V_{nm} and W_{nm} are the result of the evaluation of the Legendre polynomial of degree n and order m. V_{nm} and W_{nm} are computed recursively using an iterative procedure starting from:

$$V_{00} = \frac{R_\oplus}{r}, \quad W_{00} = 0.$$

Note that using only those values corresponding to index "00" the Earth potential is:

$$U_0 = \frac{GM_\oplus}{R_\oplus} \left(C_{00} V_{00} + S_{00} W_{00} \right) = C_{00} \frac{GM_\oplus}{r}$$

Therefore, setting $C_{00} = 1$ we get the unperturbed Newton's potential.

The iterative computations for V_{nm} and W_{nm} depends on whether $m = n$ or $m < n$. $V_{nm} = 0$ and $W_{nm} = 0$ when $m < n$. First one can compute recurrently the set of terms that $m = n$ by means of:

$$\begin{aligned} V_{mm} &= (2m-1) \left(\frac{xR_\oplus}{r^2} V_{m-1,m-1} - \frac{yR_\oplus}{r^2} W_{m-1,m-1} \right), \\ W_{mm} &= (2m-1) \left(\frac{xR_\oplus}{r^2} W_{m-1,m-1} + \frac{yR_\oplus}{r^2} V_{m-1,m-1} \right). \end{aligned}$$

Once these elements are computed we can recurrently compute the remaining ones using:

$$\begin{aligned} V_{nm} &= \left(\frac{2n-1}{n-m} \right) \frac{zR_\oplus}{r^2} V_{n-1,m} - \left(\frac{n+m-1}{n-m} \right) \frac{R_\oplus}{r^2} V_{n-2,m}, \\ W_{nm} &= \left(\frac{2n-1}{n-m} \right) \frac{zR_\oplus}{r^2} W_{n-1,m} - \left(\frac{n+m-1}{n-m} \right) \frac{R_\oplus}{r^2} W_{n-2,m}. \end{aligned}$$

With all the values obtained and taking the coefficients C_{nm} and S_{nm} from a database it is not difficult to obtain the force coming from the Earth potential 4.1.

In our research the coefficients C_{nm} and S_{nm} have been extracted from the JGM-3 (Joint Gravity Model) elaborated by the NASA's Goddard Space Flight Center in collaboration with the University of Texas Center for Space Research and the Centre National D'Études Spatiales (CNES). The adaptation of all these computations to the Jet Transport is straightforward. We only need to replace the usual double precision algebra by the polynomial algebra. Observe that all the terms V_{nm} and W_{nm} are polynomials in a Jet Transport implementation since they are combinations of x, y and z which are themselves polynomials.

4.1.2 Third body perturbations: the Moon and the Sun

Other important peturbative forces are the gravitational influences of the Sun and the Moon, one because of its dimension and the other because its proximity to the Earth. Since the Earth-Sun and the Earth-Moon distances are big enough their contribution to to the potential can be assumed of punctual masses. However, we must take into account that when we introduce them in the model the central body is also affected, so we must consider not their absolute acceleration but the relative acceleration of the spacecraft with respect to the Earth:

$$\vec{a} = \mu \left(\frac{\vec{r} - \vec{x}}{\|\vec{r} - \vec{x}\|^3} - \frac{\vec{r}}{r^3} \right).$$

Where here $\vec{r}$ is the position vector of the perturbing 3rd body, $\vec{x}$ is the position vector of the satellite and μ is the mass parameter of the 3th body, i.e. $\mu = G \cdot M_b$, where M_b is the mass of the Sun or the Moon. Note that in this expression, the first term is the influence of the third body on the satellite while the second one is the influence of the third body on the Earth. In such way we obtain the relative acceleration on the spacecraft.

Therefore, we must determine the position of both celestial bodies, Sun and Moon. This can be done in different ways considering the usual trade between accuracy and computational speed. Some of the options available are:

- To consider planar and circular orbits, i.e:

$$\begin{cases} x(t) = r\cos(\omega t) \\ y(t) = r\sin(\omega t) \\ z(t) = 0 \end{cases}$$

- To consider a low precision model expanding the longitude and latitude of the Sun and Moon as series in the orbital elements.
- To integrate the position of the Sun and the Moon together with the satellite. In this case we note that for the satellite we consider jet transport but for Sun and Moon a single trajectory is enough. For this purpose we can integrate first the position of these bodies during a time interval and to evaluate their position when needed for the jet transport of the satellite.
- To read the relative position of the Sun and the Moon with respect to the Earth from an ephemerides file.

We have selected two of the four models to compare: the low precision model and the ephemerides model. Using the model with circular and planar we have a first approximation to the position of the bodies, however it presents inaccuracies for mid term integrations. On the other side, the propagation of the full system is expensive and it does not incorporates more perturbations as the ephemerides one includes.

Low precision model

In order to define the position of the Sun and the Moon a low precision model can be used. To obtain fast approximations of their position in longitude, latitude and the distance from the Earth, we use their mean orbital elements and assume unperturbed motion. The underlying low precision model is described in [MG05]. Positions of the Sun and Moon are obtained computing the ecliptic longitude, latitude and the distance from the Earth using orbital elements as follows. For the Sun:

$$\begin{aligned} \lambda_\odot &= \Omega + \omega + M + 6992'' \sin M + 72'' \sin 2M, \\ \beta_\odot &= 0, \\ r_\odot &= (149.461 - 2.499\cos M - 0.021\cos 2M) \cdot 10^6, \end{aligned}$$

where $\Omega+\omega = 282.9400^\circ$ is the sum of the longitude of the ascending node and the argument of the periapsis (because inclination of Sun in the ecliptic is zero), $M = 357.5256^\circ + 35999.049^\circ \cdot T$ is the mean anomaly and T is the epoch measured in Julian centuries from year 2000, $T = (JD - 2451545.0)/36525.0$

Since the motion of the Moon is faster and very perturbed, more coefficients are needed:

$$\begin{aligned}
\lambda_{\,\mathbb{)}} &= L_0 + 0.109761817403199\sin(l) + 0.00372821720773233\sin(2*l) \\
&\quad -0.0222335554156833\sin(l-2*D) + 0.0114900842422960\sin(2*D) \\
&\quad -0.00323855538981170\sin(l') - 0.00199743236617129\sin(2*F) \\
&\quad -0.00102780500395222\sin(2*l-2*D) - 0.000998716183085644\sin(l+l'-2*D) \\
&\quad +0.000930842267730309\sin(l+D*2) - 0.000799942573830734\sin(l'-2*D) \\
&\quad +0.000717524248042113\sin(l-l') - 0.000606017101386920\sin(D) \\
&\quad -0.000533295049220490\sin(l+l') - 0.000266647524610245\sin(2*F-2*D), \\
\beta_{\,\mathbb{)}} &= 0.0897874937414861\sin(F+\lambda_{\,\mathbb{)}} - L_0 + 0.00199743236617129 * sin(2*F) \\
&\qquad\qquad +0.00262284201480259 * sin(l')) \\
&\quad -0.00255011996263616\sin(F-2*D) + 0.000213318019688196\sin(l+F-2*D) \\
&\quad -0.000150292241143956\sin(-l+F-2*D) - 0.000121203420277384\sin(-2*l+F) \\
&\quad -0.000111507146655193\sin(l'+F-2*D) + 0.000101810873033003\sin(-l+F) \\
&\quad +0.0000533295049220490\sin(-l'+F-2*D), \\
r_{\,\mathbb{)}} &= 385000 - 20905\cos(l) - 3699\cos(2*D-l) \\
&\quad -2956\cos(2*D) - 570\cos(2*l) + 246\cos(2*l-2*D) \\
&\quad -205\cos(l'-2*D) - 171\cos(l+2*D) - 152\cos(l+l'-2*D).
\end{aligned}$$

Here $L_0 = 3.81033597684367 + 8399.68471971156 \cdot T$ is the Moon's mean longitude, $l = 2.35554732210571 + 8328.69142518676 \cdot T$ is the Moon's mean anomaly, $l' = 6.23999591310851 + 628.301940316221 \cdot T$ is the Sun's mean anomaly, $F = 1.62791798615294 + 8433.46617912181 \cdot T$ is the mean angular distance of the Moon from the ascending node and $D = 5.19846788945409 + 7771.37714390171 \cdot T$ is the difference between the mean longitudes of the Sun and the Moon.

Therefore denoting by the sub-index b either the Moon ($b = \,\mathbb{)}$) or the Sun ($b = \odot$), their coordinates in an Earth centred ecliptical frame are:

$$\vec{r_b} = \begin{pmatrix} r_b\cos(\lambda_b)\cos(\beta_b) \\ r_b\sin(\lambda_b)\cos(\beta_b) \\ r_b\sin(\beta_b) \end{pmatrix}.$$

Finally to obtain the position of the body in an Earth centred equatorial frame we have to rotate the previous coordinates accounting for the obliquity of the ecliptic $\epsilon = 23.43929111^\circ$, i.e. multiply the vector by the matrix:

$$R_x(\epsilon) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\epsilon & -\sin\epsilon \\ 0 & \sin\epsilon & \cos\epsilon \end{pmatrix}.$$

Using this procedure we get the position of the Sun and the Moon in a fast way but with relative low precision. When more precision is required we can use more accurate ephemerides.

JPL ephemerides models

Since the time of the Earth explorations in the late middle age, the human kind has been using ephemerides to track events in the sky. Traditionally the ephemerides were transcriptions of the position of multiple stars

and celestial bodies along time collected in books. Those were crucial to determine the position of a ship in the middle of the ocean without any other reference than the starry night.

Nowadays, ephemerides are still used to locate objects in the sky and with them it is possible to know the position of celestial bodies with high precision. The DE405 ephemeris produced by the NASA's Jet Propulsory Laboratory (JPL) are used in this work. These ephemerides are given as files containing the coefficients of the Chebyshev polynomials that approximate the position of the included bodies. Chebyshev's polynomials are well known interpolating polynomials defined as:

$$T_n(x) = \cos(n \arccos x),$$

and there exist wide descriptions in the literature, for instance in [PTVF07] one can find their description and many good properties. Let us stress some of those characteristics that are important for our work. The n-th Chebyshev polynomial can be recursively defined by:

$$\begin{aligned} T_0(x) &= 1, \\ T_1(x) &= x, \\ T_2(x) &= 2x^2 + 1, \\ &\dots \\ T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x). \end{aligned}$$

Therefore, given a certain continuous function f it is possible to obtain some coefficients c_i in such a way that

$$f(x) \approx \left[\sum_{k=0}^{N} c_k T_k(x)\right] - \frac{1}{2}c_0,$$

so, f is approximated inside an interval $[a, b]$ for given set of suitable coefficients. Indeed, it is also well known that any continuous function can be approximated by polynomials and that the Chebyshev polynomials are a base of the polynomials.

In the other way round, if the Chebyshev coefficients that approximate a given function f are known, it is possible to compute the approximation of the function in a given point, using recursive formulae, but due to stability reasons it is convenient to use the polynomials in the interval $[-1, 1]$. Therefore, the point is transformed from the interval $[a, b]$ into the interval $[-1, 1]$ by the change of variables:

$$x = \frac{2x_o - a - b}{b - a},$$

where $x_o \in [a, b]$, $x \in [-1, 1]$ and the recurrences are given by:

$$\begin{aligned} d_{m+1} &= d_m = 0, \\ d_j &= 2xd_{j+1} - d_{j+2} + c_j \quad j = m-1, m-2, \dots 1, \\ f(x) &= d_0 = xd_1 - d_2 + \tfrac{1}{2}c_0. \end{aligned} \tag{4.2}$$

Figure 4.1 shows the relative differences between the position of the Sun (left) and the moon (right) using the low precision model and the ephemerides model. The differences observed are in a small range for the given time span.

How to adapt the position to a Taylor Method

To implement a Taylor integration method one has to take into account that all the variables depending in time should be expanded as well. The position of both bodies, the Sun and the Moon, depend on the time. Therefore, their time expansions should be computed. However, the difference introducing this small effect is expected to be small because the time steps needed to compute future positions of the spacecraft are small in the Sun, and even in the Moon time scale. Hence, some study about the effect of taking into account this fact should be done, considering, both, the computation time and the accuracy of the final results.

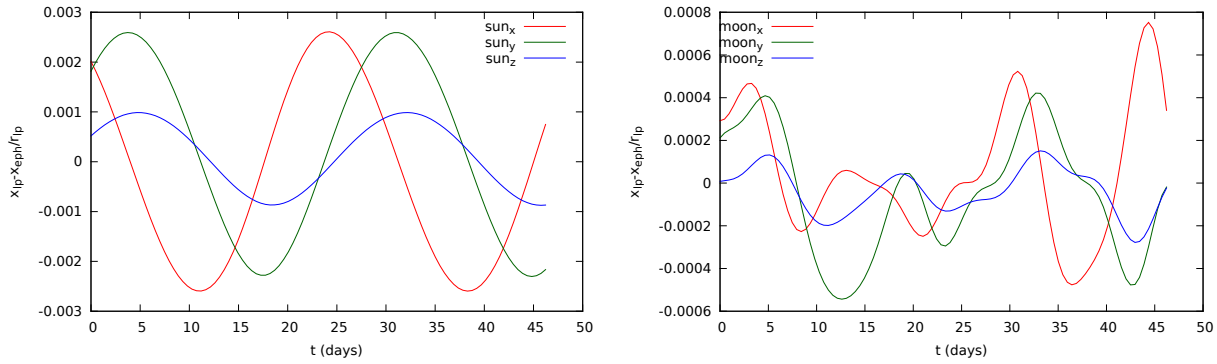


Figure 4.1: Relative errors between the Sun (left) and the Moon (right) using the low precision model and the ephemerides.

If high precision integrations are required the position of the third body as a function of time is needed. I.e. the desired coordinates of the third body must be expanded as a Taylor polynomial. This polynomial gives the current position of the third body when it is evaluated at time 0 and its position after T units of time when it is evaluated at time T.

Such polynomial can be computed using the following three steps:

- Obtain Taylor coefficients from the model selected for the third body.
- Transform this coordinates to a geocentric model.
- Introduce the Earth rotation.

The first step will work slightly different depending on the selected model. The objective for this step is to obtain the coefficients of the Taylor expansion, $\hat{x}_i$, of the function $x(t)$ representing each one of the coordinates. In the case of a low precision model we have to substitute all the values of T by $t_0 + \delta t$. Then expand successively the Taylor series of the sines and the cosines for the appropriate values in order to obtain the position of the bodies as Taylor expansions.

To obtain the Taylor expansions for the ephemerides model we need to do some minor changes to the recursive formulae in the Chebyshev evaluation. Given the Chebyshev coefficients $\overline{x}_i$ which approximate $x(t_0 + t)$, we want to compute the approximation of $x(t_0 + \tau)$ using Taylor polynomials with coefficients $\hat{x}_i$. First we proceed with the change of variables:

$$t = \frac{2(t+\tau) - t_0}{\Delta t} - 1 = \underbrace{\frac{2}{\Delta t}}_{A}\tau + \underbrace{\frac{2(t-t_0)}{\Delta t} - 1}_{B},$$

where $t_0 = a + b$ and $\Delta t = b - a$.

Now we apply (4.2) to determine the coefficients of the polynomials:

$$\begin{aligned}
&d_{m+1,k} = d_{m,k} = 0 \quad \forall k,\\
&\begin{cases} d_{j,0} = 2Bd_{j+1,0} - d_{j+2,0} + \overline{x}_i, \\ d_{j,k} = 2Ad_{j+1,k-1} + 2Bd_{j+1,k} - d_{j+2,k} \quad k > 0, \end{cases}\\
&\begin{cases} \hat{x}_0 = Bd_{1,0} - d_{2,0} + \frac{1}{2}\overline{x}_0, \\ \hat{x}_k = Ad_{1,k-1} + Bd_{1,k} - d_{j+2,k} \quad k > 0, \end{cases}
\end{aligned}$$

In such way the Taylor polynomials for the three objects $\hat{s}_i$, $\hat{m}_i$, $\hat{b}_i$ are obtained.

The last different step when we are using ephemerides consist in transforming the Taylor polynomials to geocentric coordinates (observe that in the low precision model the coordinates are already geocentric). In the DE405 the coordinates for the Moon are geocentric. Therefore, we do not need to do any additional change ($M_i = \hat{m}_i$). However, the Sun has to be transformed using the coordinates of the Earth-Moon baricenter ($\hat{b}_i$), the Moon ($\hat{m}_i$) and the ones of the Sun ($\hat{s}_i$), using the formula:

$$S_i = \hat{s}_i - \hat{b}_i + \frac{1}{1+\mu^\star}\hat{m},$$

where $\mu^\star$ is the Earth-Moon mass ratio that can be extracted from the DE405 file. We are subtracting the coordinates of the baricenter and adding the coordinates of the Earth in the initial reference system.

Finally, in both models, the Earth rotation has to be introduced. In most implementations the rotation is done to the spherical harmonics, but in the case of a Taylor implementation to transform all the harmonics would become too expensive while to transform just the perturbations due to the third bodies is made using some products with cosines and sines. In this final step, we define the expansions for the cosines and the sinus as:

$$cs_i = \frac{\omega^i}{i!}\begin{cases}\cos(\omega t) & i = 0 \pmod 4\\ -\sin(\omega t) & i = 1 \pmod 4\\ -\cos(\omega t) & i = 2 \pmod 4\\ \sin(\omega t) & i = 3 \pmod 4\end{cases}; \quad si_i = \frac{\omega^i}{i!}\begin{cases}\sin(\omega t) & i = 0 \pmod 4\\ \cos(\omega t) & i = 1 \pmod 4\\ -\sin(\omega t) & i = 2 \pmod 4\\ -\cos(\omega t) & i = 3 \pmod 4\end{cases}.$$

Then we apply the rotation of angle ωt (or $\omega t + \varphi$ if an additional phase between the body and the Earth is taken into account) computing the polynomial products with the obtained series.

$$\begin{aligned} s_x &= \cos(\omega t) * S_x + \sin(\omega t) * S_y,\\ s_y &= -\sin(\omega t) * S_x + \cos(\omega t) * S_y. \end{aligned}$$

Where $S = (S_x, S_y, S_z)$ are the coordinates of the Sun. Therefore the general formula for this rotation should be:

$$\begin{aligned} s_{x,j} &= \sum_{i=0}^{j} cs_i \cdot S_{x,j-i} + \sum_{i=0}^{j} sn_i \cdot S_{y,j-i},\\ s_{y,j} &= -\sum_{i=0}^{j} sn_i \cdot S_{x,j-i} + \sum_{i=0}^{j} cs_i \cdot S_{y,j-i}. \end{aligned}$$

Analogously we proceed for the coordinates of the moon.

At the end of the computation we obtain polynomials determining the position of the Sun ($s = (s_x, s_y, s_z)$) and the Moon ($m = (m_x, m_y, m_z)$).

The usage of Taylor series to compute the position of the perturbing bodies increases the accuracy of our computation. As a drawback it slows down the procedure and makes it more CPU consuming. The effect of not using the Taylor series can be reproduced in a RK scheme by fixing the third body in a certain position for a given time. For instance, assuming that the Moon moves in a discrete way once every a-dimensional unit of time. Figure 4.2 shows the difference in the first coordinate of a spacecraft taking into account in one case that the third bodies are in its correct place (true positions) and in the other one that the body is moving once every time unit using both, the ephemerides and the low precision models.

Figure 4.3 shows the error that we obtain when we integrate the system using a Taylor series for the position of the third body or we just fix it for all the time steps. Observe that, as it is expected, the results are similar to the ones obtained in Figure 4.2, where a RK method is used.

4.1.3 Atmospheric drag

Atmospheric drag is the force resulting from the friction between the atmosphere particles and the spacecraft. Since it is caused by the atmosphere it mainly affects spacecraft in low Earth orbits. Its direction is opposite to the velocity and modelled by:

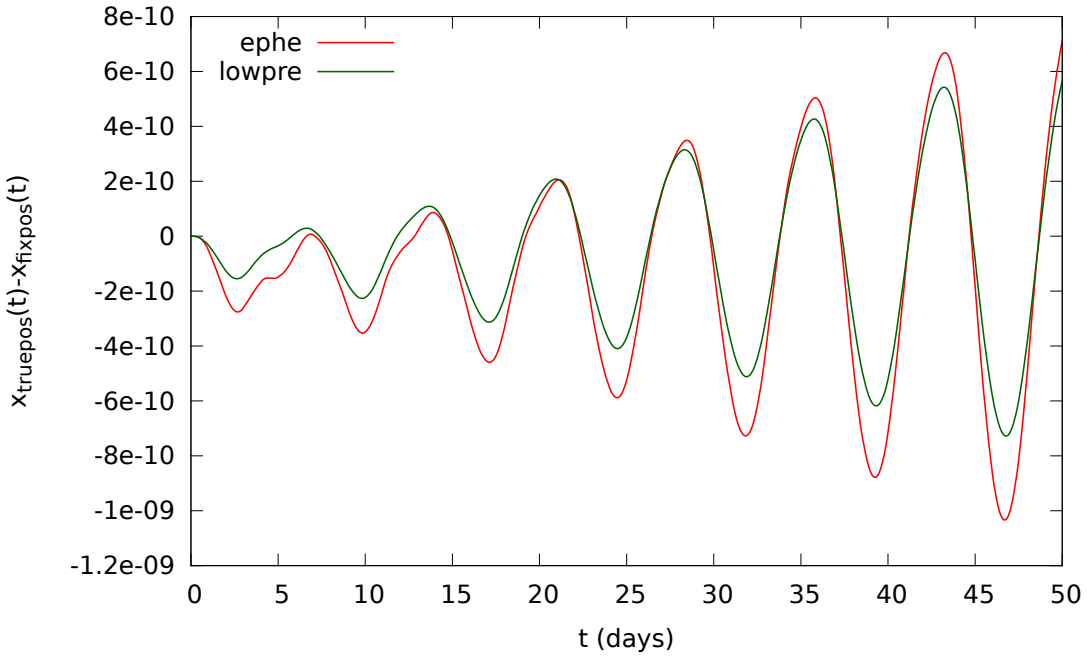


Figure 4.2: Difference in the first coordinate of a spacecraft when the 3rd body is moving in a discrete way (i.e. the 3rd body is fixed for 1 time unit) and when it moves in a continuous way for the ephemerides model (red) and the low precision (green).

$$\vec{a} = -\frac{1}{2}\rho(\vec{x})\underbrace{\frac{S\ C_N}{M}}_{B_c}\ v_r^2\ \frac{\vec{v}_r}{v_r} = -\frac{1}{2}\ \rho(\vec{x})\ B_c\ v_r\ \vec{v}_r,$$

where $\rho(\vec{x})$ is the atmospheric density at $\vec{x}$, S is the sphere surface, C_N is the aerodynamic coefficient and M is the Mass of the satellite. Finally v_r is the velocity of the satellite with respect to the atmosphere. Observe that the atmosphere is rotating in our coordinate frame. Hence the velocity of the atmosphere must be subtracted to the velocity of the spacecraft v. The atmosphere is moving with a velocity perpendicular to the position of the satellite with modulus Ω where Ω is the the angular velocity of the planet. Therefore, the velocity of the atmosphere is $v_a = (\Omega y, -\Omega x, \Omega z)$, where (x, y, z) is the current position of the spacecraft. Then, the final relative velocity is

$$v = (\dot{x} + \Omega y, \dot{y} - \Omega x, \dot{z}).$$

Finally, the parameter B_c is the ballistic coefficient of the sphere, and is determined as $B_c = \frac{S\ C_N}{M}$. In some cases this parameter is supplied instead of the sphere surface, the aerodynamic coefficient and the mass of the satellite. The only term that remains to be known is the atmospheric density and for this purpose there exist also several models.

Atmospheric density models

To model the atmosphere is a complex task for real applications although there exist several options in the literature. Some models assume that the density is height constant, then a formula is provided to compute the density at each height. Others take into account also several factors, like the incident solar-flux, the composition of the atmosphere or the geomagnetic activity.

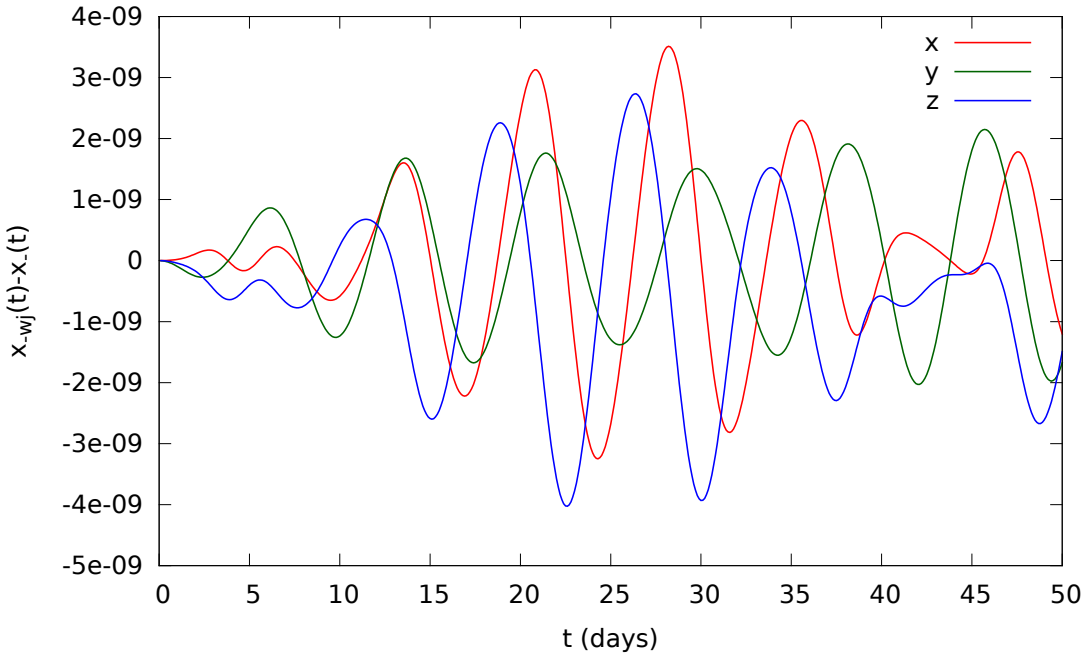


Figure 4.3: Under the Earth gravitational field and the perturbations produced by the Sun and the Moon, difference between the position of the spacecraft computed fixing the third bodies during each step and computing its Taylor series in the x (red), y (green) and z (blue) components.

The main model selected for this dissertation is the exponential model. It gives the density as a function of the height of the satellite and some parameters that must be fixed. However, it does not take into account other complex factors like the ones mentioned before, at a given height it only accounts for most abundant molecule and the mean temperature. The MSIS-E-90 model has been considered as a one that incorporates complex factors of the problem. However, the translation of this model into the Jet transport is not an straightforward task and its use has been left for future studies.

- The exponential model give the density as:

$$\rho(h) = \rho_0 \exp\left(\frac{-h}{H}\right)$$

  where:

  - h is the height of the satellite over the see level, $h = \sqrt{x^2 + y^2 + z^2} - 1$
  - $H = \frac{mg}{KT}$ is the scale height,
    - m is the mass of the most abundant molecule. From 165 to 600 km the most abundant molecule is oxygen, O_2, Between 550 and 900 km it is helium, H_e and above it is hydrogen, H. Their respective masses are $32N_A$, $2N_A$ and $1N_A$, where $N_A = 6.022 \cdot 10^{23}\text{mol}^{-1}$ is the Avogadro constant.
    - g is the gravity parameter. For satellites around the Earth it can be approximated by $g = -9.8\frac{m}{s^2}$.
    - K is the Boltzmann constant ($1.3806488 \cdot 10^{23}\frac{m^2 Kg}{s^2 K}$).

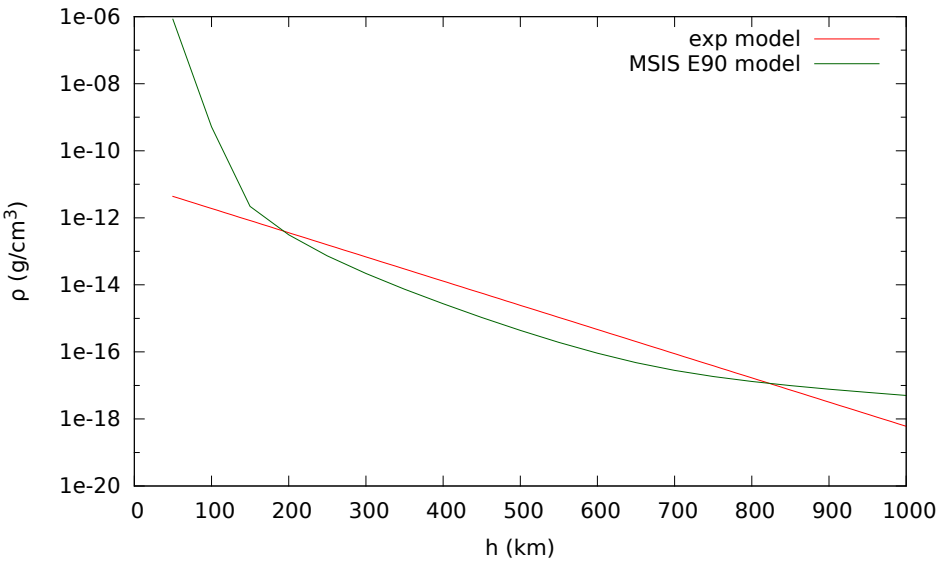


Figure 4.4: Value of the atmospheric density as a function of the height for the exponential model (red) and the MSIS model on January 1st 2000 (green).

    * T is the temperature. It shows high difference between the day and the night and the solar activity. Over the 300 km it takes values around 700 and 1500 Kelvin with an average value of 900 Kelvin. The average value has been taken into account for the present project.

  - ρ_0 is the density at the Earth's surface.

- Real data models: The second block of models that can be used to model the atmospheric density is based in real data and complex observations. The MSIS-E-90 atmospheric model gives the density and the temperature for different height and locations according to several parameters. In addition it uses several measurements to improve the former models at different atmosphere levels. It is also uses averaged meteorological measurements for a better knowledge of the atmosphere. Some preliminary tests have been done but due to the complexity and the parameters that includes, a full conversion to Jet transfer computations in polynomial algebra has not been completed. For quick tests it is possible to evaluate density models as a function of the height in the webpage `http://ccmc.gsfc.nasa.gov/modelweb/atmos/msise.html`. It is also possible to download the code from the same source.

  There exist also other models for realistic data as it can be Jacchia or the ones used by the ESA (Earth-GRAM or CIRA). Further studies including these models have been considered for a future research.

Anyway, a comparison between the exponential model and the MSIS model has been done in order to check that the density is accurately determined. Figure 4.4 shows the density computed with the exponential model, using an averaged molecule as most abundant molecule, and the MSIS model on January 1st 2000. We can see that the exponential model taken is acceptable for heights over 150 km but under this height the approximation is not good enough. That fact is due to a change in the most abundant molecule, below 165 km the most abundant molecule is nitrogen N and that has not been introduced in the average of molecules. For the low Earth atmosphere another exponential model should be considered, but the height of the studied satellites will always be over 165 km and hence this approximation is good enough.

4.1.4 Tests on the models

Let us now present the test of the different the models and their implementations. The tests are divided in two parts. The first one consists in checking the precision obtained by the method considering the use of different orders and degrees for the model as well as for the integration method. Then the result obtained is compared with a RK-78 integration. An special care is taken when checking that the propagations made using the evaluation of the Jets correctly approximates the flow at time T. Finally a test on the size of the neighbourhood around the central point, $\vec{x}_0$, is done. The second part of the section is devoted to the study of the time overload of different perturbations, and to study when is worth to use a Jet implementation for Monte Carlo analysis. All the computations have been made in an Intel Xeon E5645 (2.40Ghz, 12Mb Cache and 5.86GT/s).

Accuracy checks

First we check the accuracy of the Jet implementation in both, first order terms and higher order terms. For this purpose random initial conditions are selected as follows:

- The initial position (x, y, z) is randomly selected at a distance between one and two Earth radii from the centre of the Earth. The samples follow a uniform distribution.
- The initial velocity $(\dot{x}, \dot{y}, \dot{z})$ is selected perpendicular to the radius and with a modulus between 7 and 8 km/s also with an uniform random distribution. Using this velocity we ensure that the orbit of the spacecraft does not go close to the singular point, the centre of the planet.

The initial Keplerian energy for this initial conditions is in the range $[-0.6, 0)$. This gives us a parameter to classify the different orbits. Note that, the smaller the Keplerian energy, the closer to the centre of the Earth the orbit goes (when the orbit has close passages to the centre of the Earth the integration procedure needs of more steps and, time consumption increases). The model implemented for this first test is the model 1010, i.e. we use the Earth potential of order and degree 10 without including any other perturbation. Figure 4.5 shows the errors between a Jet-Taylor integration of order 20 and the RK-78 for one day, one week and one month. The longer the integration time the bigger are the errors, as it is expected. However, they still are in an acceptable range.

The second test consists in analysing higher order variational terms. As we saw in Chapter 2, higher order variational terms can be checked using the previous orders. The order zero is already checked using the RK-78. Using those terms we can approximate the first variational terms by means of numerical differentiation,

$$v_{ij} = V_j^{(x_i)}(t; t_0, \vec{x}_0) = \frac{\varphi_j(t; t_0, \vec{x}_0 + \delta e_i) - \varphi_j(t; t_0, \vec{x}_0 - \delta e_i)}{2\delta},$$

where (v_{ij}) is the State Transition Matrix. More generally, for the higher order terms we can also use the previous order to numerically evaluate the result:

$$V_j^{(x_1^{\alpha_1}, \ldots, x_n^{\alpha_n})}(t; t_0, \vec{x}_0) = \frac{V_j^{(x_1^{\alpha_1}, \ldots, x_n^{\alpha_n})-e_k}(t; t_0, \vec{x}_0 + \delta e_k) - V_j^{(x_1^{\alpha_1}, \ldots, x_n^{\alpha_n})-e_k}(t; t_0, \vec{x}_0 - \delta e_k)}{2\delta}.$$

Note that the value of $V_j^{(x_1^{\alpha_1}, \ldots, x_n^{\alpha_n})-e_k}$ is in the previous order of the Jet and therefore we can easily implement the scheme.

We remark that automatic differentiation formulas are usually ill conditioned. Although we are using the formulas to compute first derivatives we must take care of the propagation of the errors, since in the checks we are using values that can not be accurate. It is possible to implement the same Richardson extrapolation scheme that we developed in the CRTBP in Chapter 2. Figure 4.6 shows the differences between the numerical differentiation using different values of δ and the value of the variational terms using the Jet. It can be observed that the smaller the value of δ is, the smaller is the difference. However, the difference between using $\delta = 10^{-4}$ and $\delta = 10^{-6}$ is not noticeable. This happens because round off errors are bigger than the errors of the numerical formula.

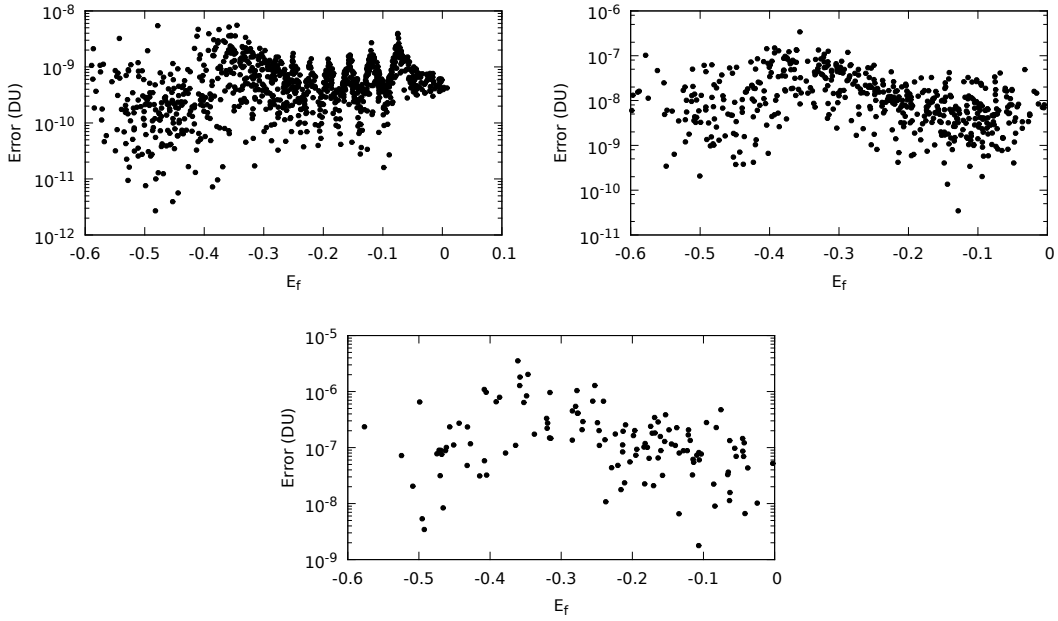


Figure 4.5: For $t = 1$ day, 1 week and 1 month, differences between the RK78 and a jet propagation Taylor method of order 10 as a function of the final Keplerian energy (E_f). The differences are given in canonical units. The order and the degree of the of the gravitational Earth potential is 10.

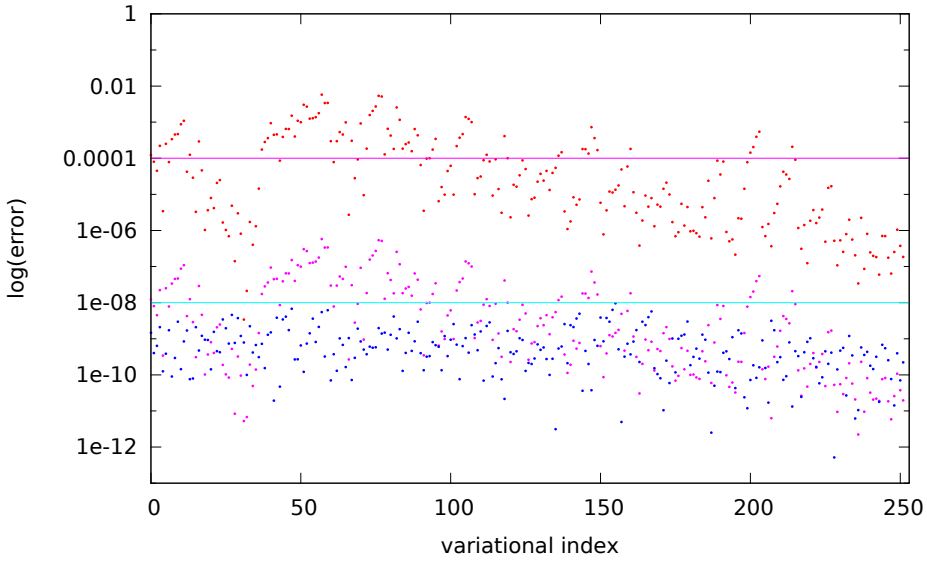


Figure 4.6: Differences between the variational equations for orders 1 and 2 and the Jet computed using numerical derivatives with some differentiation steps ($\delta = 10^{-2}, 10^{-4}, 10^{-6}$ respectively for red, purple and blue).

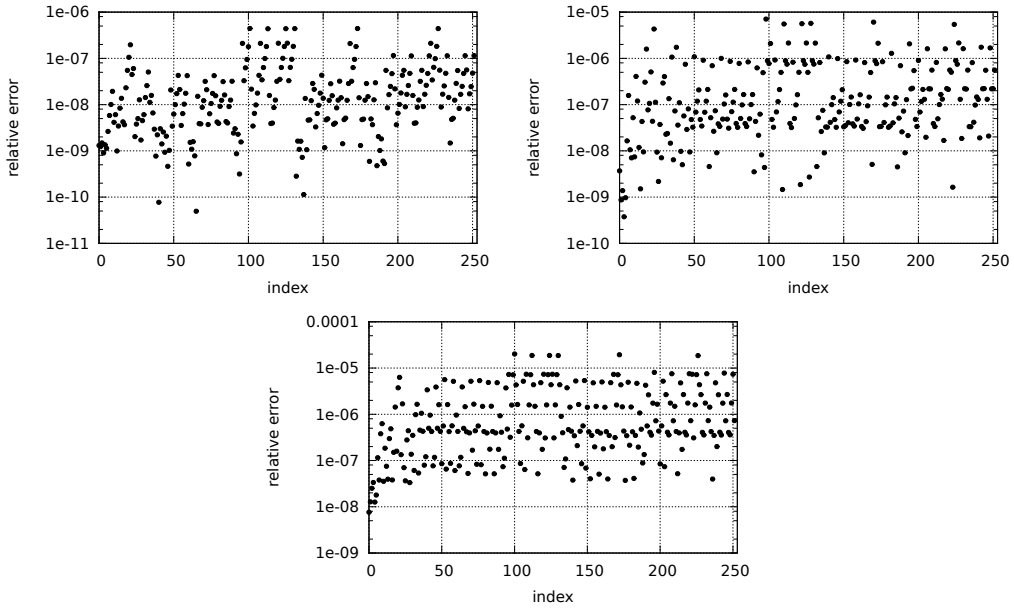


Figure 4.7: For $t = 1$ day, 1 week and 1 month, relative errors of the higher order terms of the jet as a function of their index (the first 36 terms correspond to the first order variational equations, the next 216 to the second order variational equations). The errors are computed as the difference between the coefficient c_k of the jet and the corresponding numerical derivative computed with the preceding order. The order and degree of the gravitational Earth potential is 5. The Taylor model is of order 20.

Figure 4.7 shows the differences in the variational terms using the same procedure for different final times: 1 day, 1 week and 1 month. As it happens with the usual propagations, the errors are bigger for longer integrations.

Figure 4.8 shows the accuracy of the integration using different order combinations in both, the Taylor method and the vector field. We assumed as correct orbit the one computed using order 20 in the Taylor's method and order and degree 16 in the full Earth potential. The result of the integration using lower orders is compared with the previous one. We can see that in terms of accuracy, the most important fact is the order in the potential. The order in the Taylor's method gives only small differences between the computations performed with the same potential and can be observed using the same order and degree as in the reference orbit. The errors produced by lower order of the potential are due to the fact that we are using different vector fields and not to the integration method.

The same kind of computations can be analysed for a given neighbourhood U of the point $\vec{x}_0$ instead of taking just the central point $\vec{x}_0$. We consider an initial condition $\vec{x}_0$. For a given final time, T_f, compute the polynomial $P^1_{T_f,\vec{x}_0}$ and $P^2_{T_f,\vec{x}_0}$ using two different combinations of order and degree of the Earth potential and order of the Taylor's method. We obtain random uniformly distributed variables around $\vec{x}_0$ given by the deviation $\vec{\xi}$. Finally, we compute the difference between the two propagations $P^1_{T_f,\vec{x}_0}(\vec{\xi})$ and $P^2_{T_f,\vec{x}_0}(\vec{\xi})$ to compare both results.

Using this sort of experiments we are interested in seeing the effect of modifying one of the variables that defines the procedure: the order of the Taylor's method and the order and degree of the Earth potential. For this reason we fix the order of the Taylor's method and we change the order and degree of the Earth potential to study the second one and vice-versa. Figure 4.9 shows the difference computed in this way between the propagation of the samples around $\vec{x}_0$ using different orders of the potential (orders 4 and 16). The differences are given as a function of the initial distance in the configuration space and the green line is the error of the central point ($\vec{x}_0$). We can see that the error is essentially given by the propagation of the

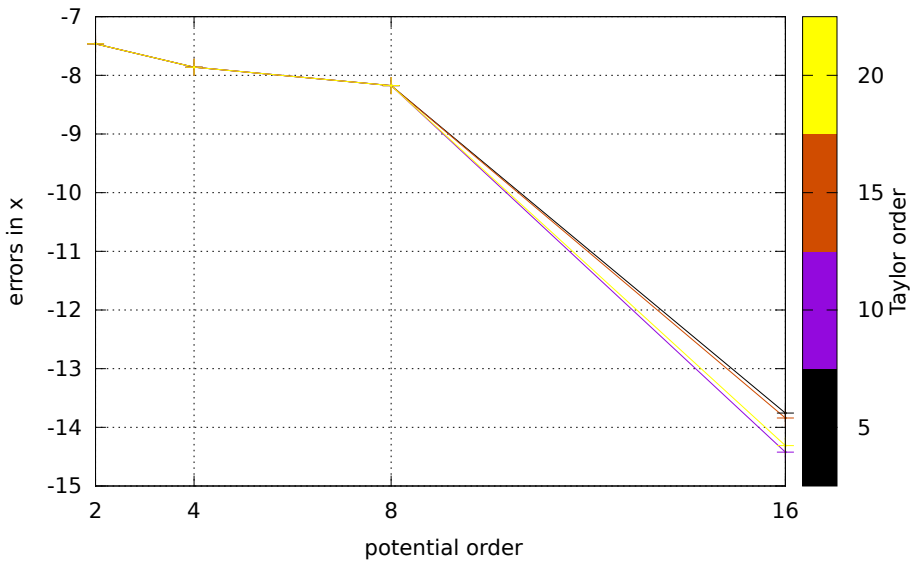


Figure 4.8: Decimal logarithm of the errors in the first configuration space variable (x) for different orders and degrees of the potential and of Taylor's method.

central point. The differences in the neighbouring points are deviations of this one. We cannot expect better propagations than the central box, but with this test we can check that there are not big discrepancies using the Jet Transport with Taylor propagations in this case.

Figure 4.10 gives similar plots but now changing the order of the Taylor method. The order and degree of the potential of the Earth has been fixed at 16 and we propagate two polynomials, $P^1_{T_f,\vec{x}_0}$ and $P^2_{T_f,\vec{x}_0}$, using order 10 and 20 for the Taylor's method respectively. As it happens in the previous case, the error of the central point, $\vec{x}_0$, (shown by the green line) dominates the errors of the neighbouring points. In this case, the difference between both polynomials is of the order of the computer accuracy.

Until now, all the tests performed study which is the effect of using different orders and degrees to integrate the system. In the following we are going to compare the Jet Transport with a Taylor integration method or with a classical integration method like the RK-78.

To compare Taylor against RK-78 performance, let us take an initial condition, $\vec{x}_0$, following an almost circular orbit with a semi-major axis $a = 7160$km and eccentricity $e = 0.001$. This approximately matches the spacecraft conditions of the Iridium constellation. Then we propagate this initial condition using the Jet Transport Taylor's method up to a final time T_f. After the propagation we get the polynomial $P_{T_f,\vec{x}_0}$ that propagates an initial condition $\vec{x}_0 + \vec{\xi}$ from the initial time $T = 0$ to the final one $T = T_f$. Once the polynomial is computed we take samples following a probability distribution around $\vec{x}_0$. Two random distributions are considered for the purpose: uniform and normal. Then, for each sample we compute the propagation of $\vec{x}_0 + \vec{\xi}_i$ using both the RK-78 procedure and the polynomial $P_{T_f,\vec{x}_0}$. Finally, we compare them computing the differences.

Considering 1000 samples, Table 4.2 shows the mean of the difference in the propagation up to 100 TU between the RK-78 and Jet Taylor method of order 15. The samples have an initial deviation with respect of $\vec{x}_0$ of at most 10^{-3} Earth radii, i.e. $\|\vec{\xi}\| < 10^{-3}$. The experiment is repeated for two different models, the first one only takes into account the effect of J_2, while the second one is an order 10, degree 10 model for the Earth potential vector-field. We can observe that the differences using both perturbative models are of the order of 10^{-3} in both, configuration and velocity space.

Figure 4.12 shows the difference between Jet Taylor and RK78 integrations as a function of the initial distance to the central point ($\vec{x}_0$) for four different models. The first model is an Earth potential with the

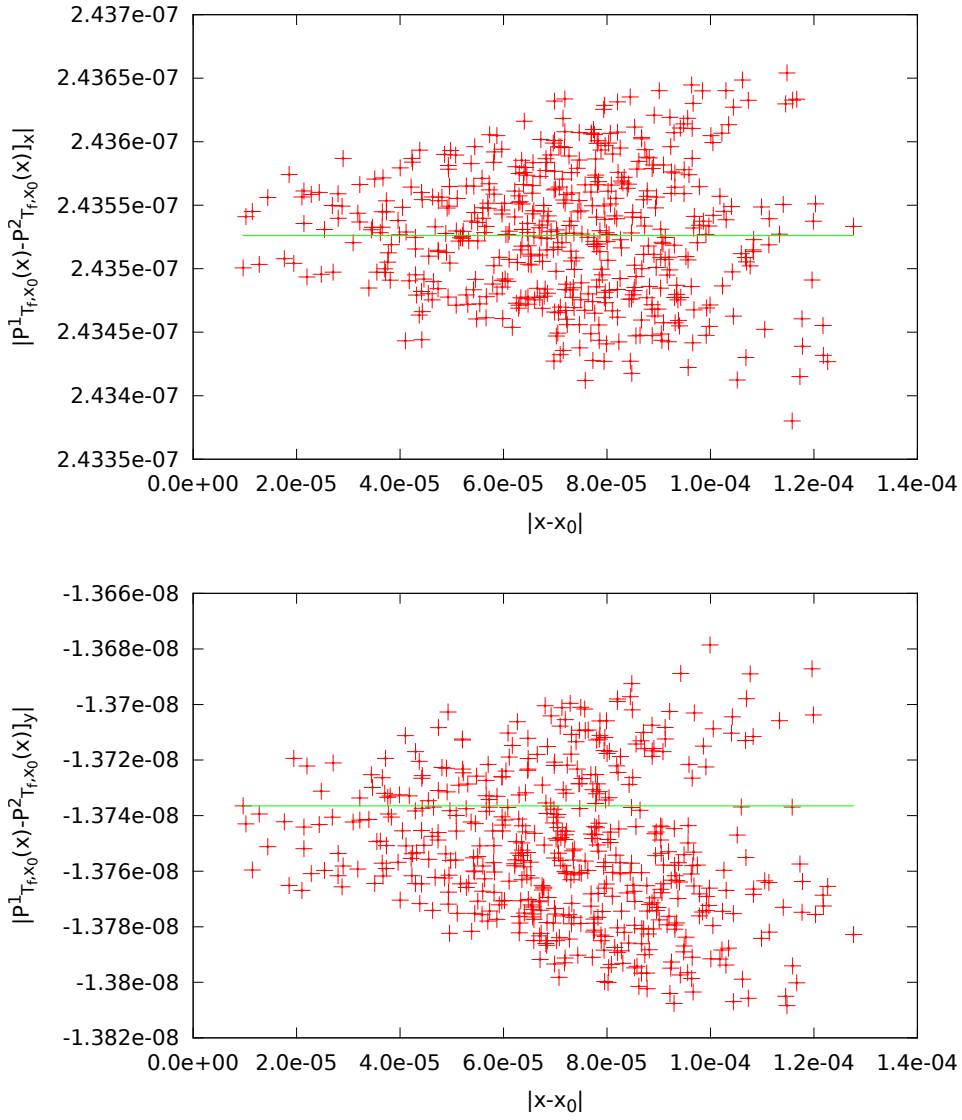


Figure 4.9: Red: Differences in the x (top) and y (bottom) variables for the propagation of an initial neighbourhood using two polynomials: $P^1_{T_f,\vec{x}_0}(\vec{\xi})$ computed using order 20 in the Taylor's method and order 4 in the Earth potential and $P^2_{t_f,\vec{x}_0}(\vec{\xi})$ computed using order 20 in the Taylor's method and order 16 in the Earth potential. Green: the error of the propagation of the central point of the neighbourhood using the same methods.

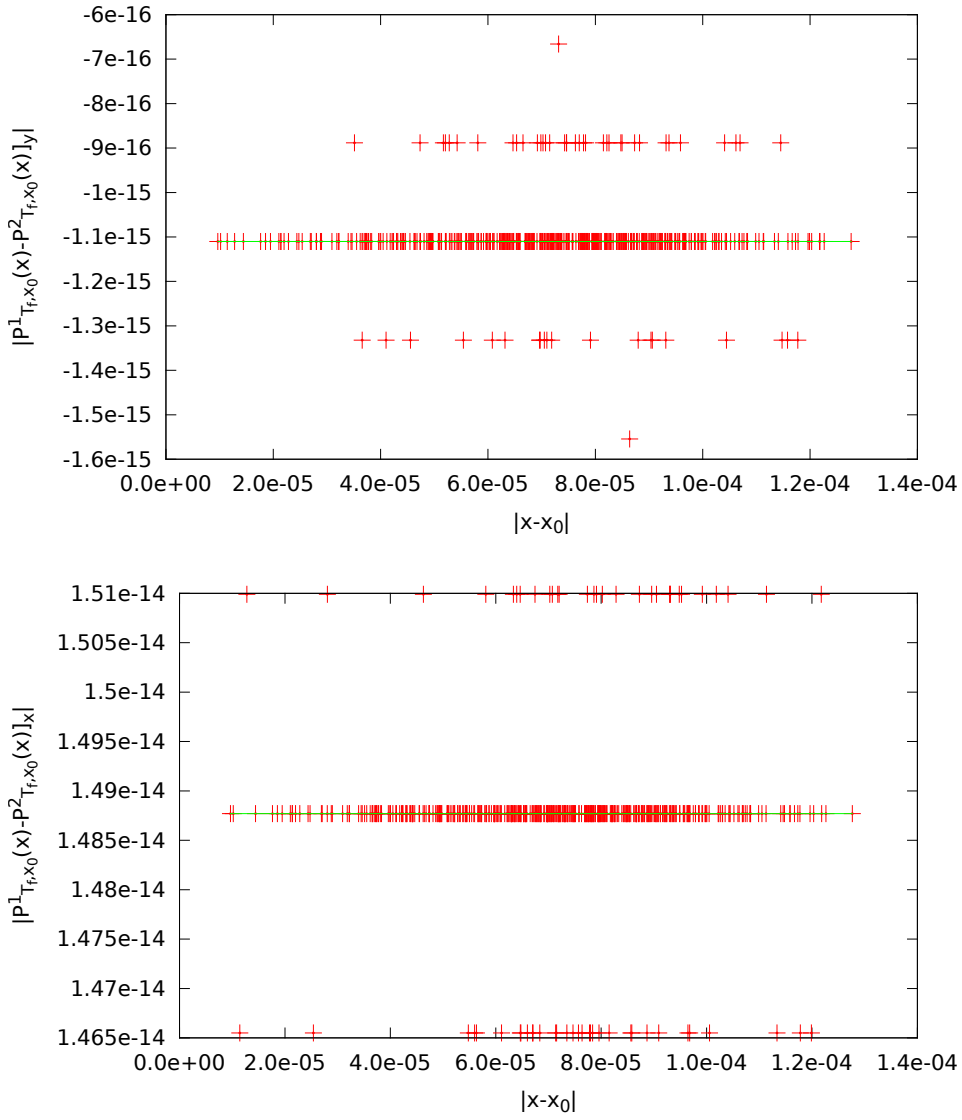


Figure 4.10: Red: Differences in the x (top) and y (bottom) variables for the propagation of an initial neighbourhood using two polynomials: $P^1_{T_f,\vec{x}_0}(\vec{\xi})$ computed using order 10 in the Taylor's method and order 16 in the Earth potential and $P^2_{t_f,\vec{x}_0}(\vec{\xi})$ computed using order 20 in the Taylor's method and order 16 in the Earth potential. Green: the error of the propagation of the central point of the neighbourhood using the same methods.

model	mean 0200 (conf)	mean 0200 (vel)	mean 1010 (conf)	mean 1010 (vel)
$\emptyset$	1.088742e-03	8.012684e-04	1.088684e-03	8.012411e-04
d	1.084485e-03	8.052435e-04	1.084422e-03	8.052207e-04
sl	1.092377e-03	1.110295e-03	1.092907e-03	7.990349e-04
sld	1.088090e-03	8.027649e-04	1.088609e-03	8.022038e-04

Table 4.2: For 1000 samples in a neighbourhood of size 10^{-3} DU, we show the mean difference between the propagation up to time $T_f = 100$ using the RK-78 method and the polynomial $P_{T_f,\vec{x}_0}(\vec{\xi})$ computed using the Jet Taylor method. The errors are splitted in configuration (conf) and velocity (vel) spaces. Two different models are considered: a model taking into account the J_2 term (left) and a precise Earth potential of order 10 and degree 10 (right).

effect of J_2 and without any other perturbation. The other three models add to the first one, the drag perturbation, the lunisolar perturbation and both perturbations respectively. The initial perturbations $(\vec{\xi})$ are considered in the configuration space (red) and in the velocity space (green).

The results for the four models are similar. Surprisingly, the error does not increase with respect to the initial deviation from $\vec{x}_0$. This someway suggest that there are directions where the propagation computed using the Jet Taylor method is worst than in others and it can be seen introducing a change of variables. Instead of using the Cartesian coordinates we can use a coordinate system adapted to the orbit. Let u, v, w be the basis of the reference system defined by the velocity vector, the position vector and the angular momentum respectively (as it is shown in Figure 4.11).

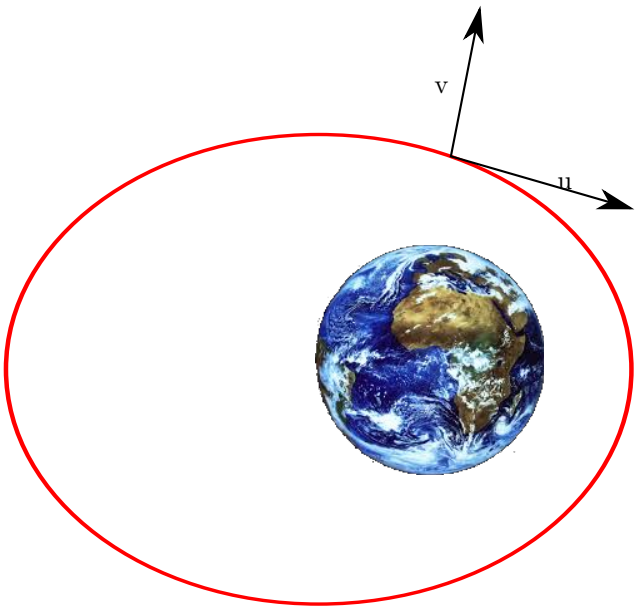


Figure 4.11: Representation of the adapted coordinates to the satellite position. The coordinate u is given by the velocity of the satellite, v by the position vector and w (not represented in the figure) completes the coordinate system.

Figure 4.13 shows the difference between propagations done using the Jet Taylor and the RK78 methods using initial deviations $\vec{\xi}$ in the directions u, v and w, and $\dot{u}$, $\dot{v}$ and $\dot{w}$. There are two directions (v and $\dot{u}$) for which, when the sample is aligned in these directions, the propagated result is worse. These two directions are the ones that modify the period of the orbit and are more sensitive to the propagation of the Jet. From

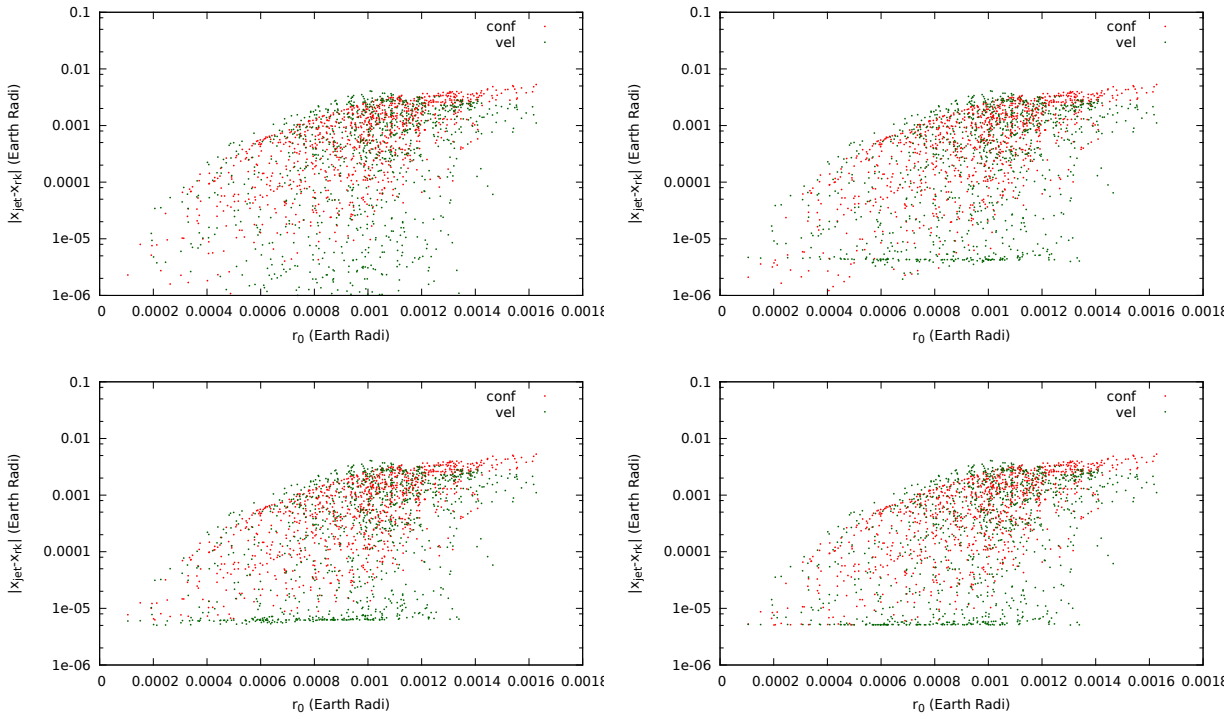


Figure 4.12: Differences between the RK78 and the Jet Taylor for different samplings around $\vec{x}_0$. The x axis corresponds to the distance from the sample to the initial central point $\vec{x}_0$. The vector field considered takes into account the J_2 term in the Earth potential in the four cases. From left to right and from top to bottom it also contains: no further perturbation, atmospheric drag, lunisolar perturbation and both lunisolar and drag perturbations. The initial conditions are considered modifying the velocity (green) and the configuration (red) variables. The final integration time is $T_f = 100$TU.

Figure 4.13 we can also see that the error is never smaller than 10^{-5}. This happens because the difference between the integration of the central point $\vec{x}_0$ using both methods (RK and Jet Taylor) is of this magnitude. Therefore we can not expect smaller differences.

Up to here we have analysed the effect of considering different orders and degrees for the Earth gravitational potential and for the Taylor method, let us now analyse in detail the impact with the order of the jet considered even that it is clear that increasing the order of the Jet the approximations should improve. Figure 4.14 shows the ellipsoids obtained after the propagation of an initial box of 64 km (in position) and 80 m/s (in velocity) for an initial elliptic orbit of semi-major axis $a = 22\ 420$ km and eccentricity $e = 0.68$ after 200 minutes for different orders of the Jet. The plot on the left shows the Jet propagation using order 1 and order 2 jets. In the order 1 jet we get the linear propagation that alternatively it could be obtained using the state transition matrix. The plot on the right shows the propagations using jets of orders 2 and 4. We can appreciate that the linear propagation coincides with the propagation of order 2 in a small region around the centre of the propagation. However, for points farther away the regions are not close at all while differences between jets of orders 2 and 4 are smaller.

Of course high order jets result in better accuracies but also imply longer CPU times of computations. A way to deal with this issue is to use higher order jets to warranty a certain accuracy at the end of the integration. For this purpose we can use the maximum initial box we introduced in chapter 2 that provides a good estimation of the maximum radius of the neighbourhood that we can propagate. Figure 4.15 shows, for an elliptic orbit ($a = 22\ 420$ and $e = 0.68$), the evolution in time of the maximum initial box that can

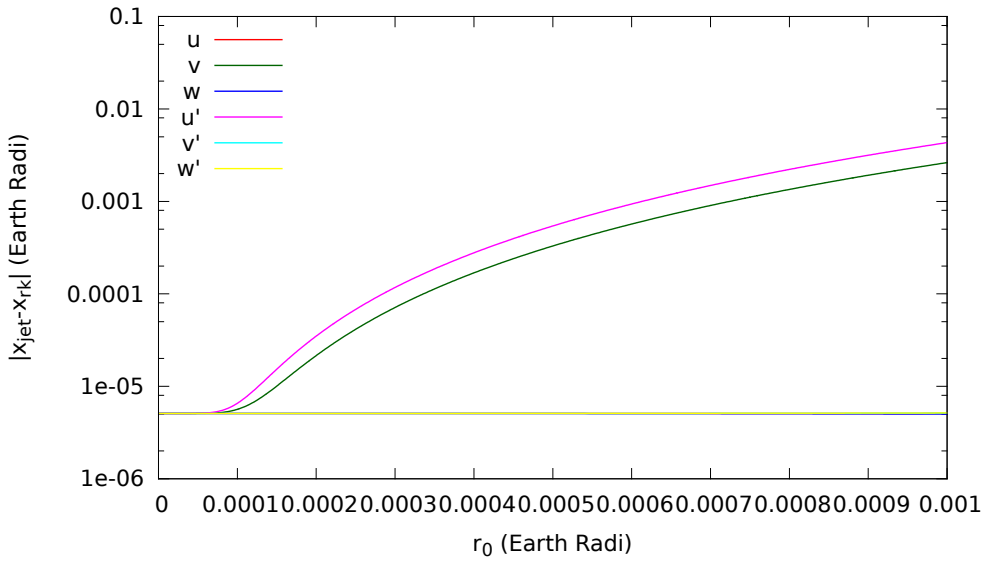


Figure 4.13: Differences between the RK78 and the Jet Taylor method for samples around $\vec{x}_0$. The x axis corresponds to the distance from the sample to the initial central point $\vec{x}_0$. The vector field considered takes into account: order 10 and degree 10 for the Earth gravitational potential and with lunisolar perturbations and drag. The initial conditions are considered modifying only one direction on the variables (u, v, w) (direction of the velocity of the spacecraft, position of the spacecraft, orthogonal direction to the previous ones) and the velocities of the spacecraft in these directions $(\dot{u}, \dot{v}, \dot{w})$. The final integration time is $T = 100$TU.

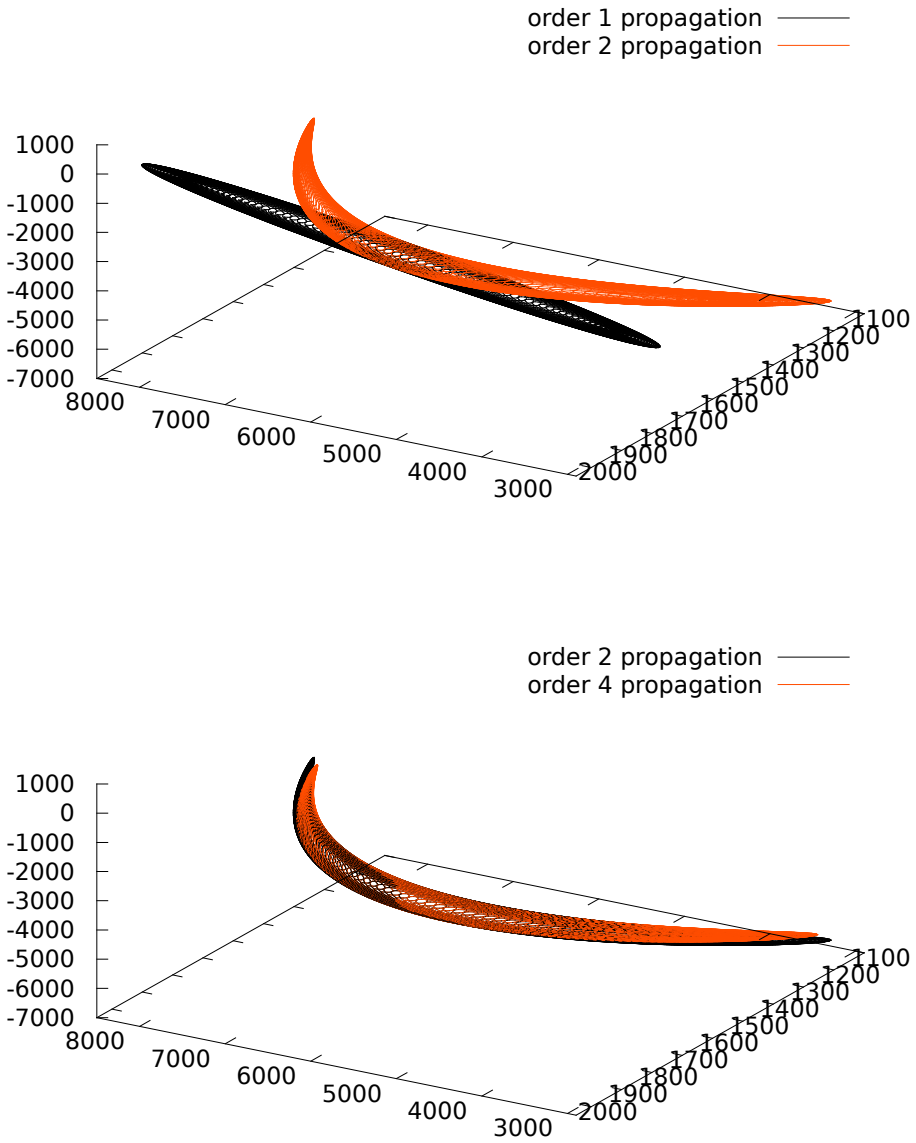


Figure 4.14: For the elliptic orbit around the Earth we plot the ellipsoid like regions obtained using jet propagations of an initial neighbourhood of 64 km size (in pos.) and 80 m/s (in vel.) for an integration time of 200 minutes. The plot on the left shows the results of the jets of order 1 (black) and 2 (orange) while on the right we have orders 2 (black) and 4 (orange).

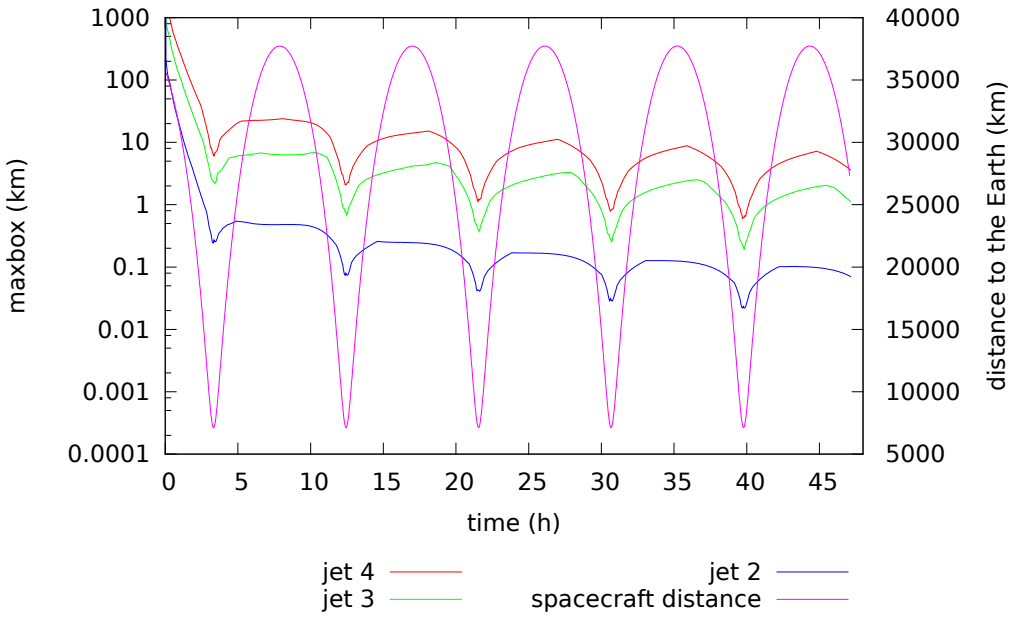


Figure 4.15: For an elliptic orbit around the Earth ($a = 22\ 420$ and $e = 0.68$) shows the maximum initial box using a jet of order 2 (blue), 3 (green) and 4 (red). The distance to the centre of the Earth is plotted in purple in the right axis.

be propagated using jets of orders 2, 3 and 4. It also shows the distance of the spacecraft to the centre of the Earth. In the three cases the maximum initial boxes that can be propagated have decreasing trend. However, the decreasing is not monotonous. When the orbit passes through the periapsis the maximum initial box decreases, when it passes through the apoapsis the maximum initial box slightly increases. This is observed at each revolution. This happens because at the periapsis the spacecraft moves faster and the polynomials do not approximate the position of it as well as when the spacecraft is moving slowly in the apoapsis.

CPU time analysis

Another fact that must be taken into account is how the inclusion of different perturbations affects the total CPU time consumption and this is important when we must decide if a jet transport implementation is convenient or not. Figure 4.16 shows the CPU time required to propagate the same initial conditions that were used in Figure 4.5. The results are again plotted as a function of the final Keplerian energy. The longer CPU times are related with the lowest values of the energy and this fact is easily explained because samples with low energy produce close orbits to the Earth and so, close to the singularity of the vector-field. In this region any integration method requires small time steps to propagate accurately enough.

Table 4.3 shows the CPU time needed to integrate, up to a final time of 2 TU, a fixed initial condition using different combinations of orders of the Taylor's method and orders and degrees of the vector-field. Of course, the CPU time and the accuracy obtained is higher when we use high orders and degrees for the vector-field. However, the order considered in the Taylor method does not increase the CPU time in the same way. One could expect that low orders in the Taylor method would give faster and less accurate integrations, but it is not the case. Lower order values in the Taylor integration give longer CPU times because the step size required to maintain the desired accuracy is very small. Therefore, the time saved at each step is fully compensated by the amount of steps required and as a result, the integrations obtained by low order Taylor methods are very long in terms of CPU time. This is clearly illustrated in the last column of Table 4.3

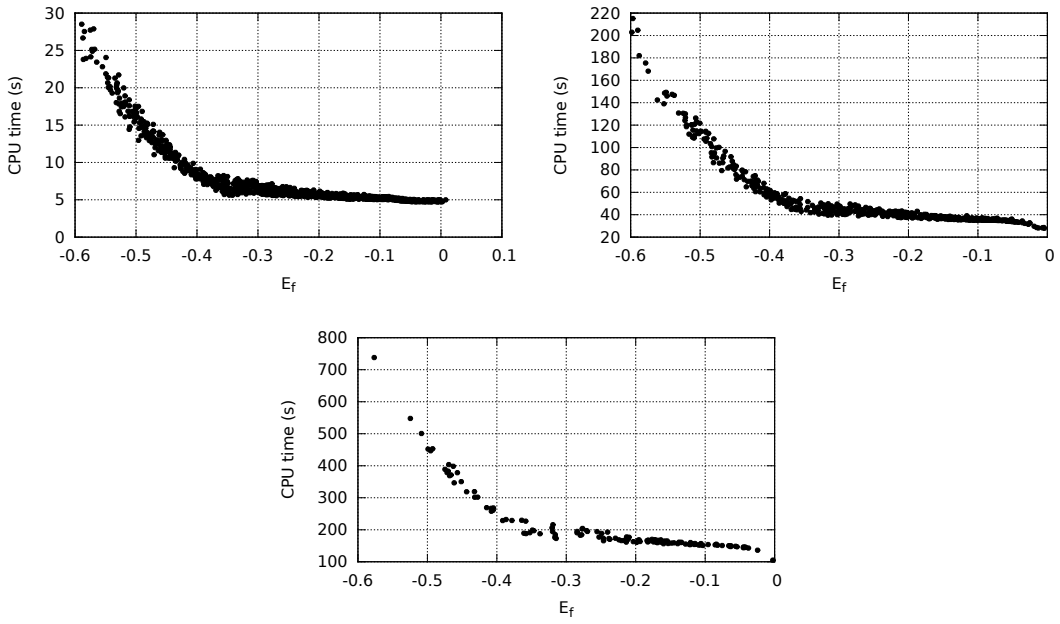


Figure 4.16: For $t = 1$ day, 1 week and 1 month, CPU time required for the determination of the polynomial $P_{t,\vec{x}_0}(\vec{\xi})$ as a function of the final Keplerian energy (E_f). The gravitational Earth potential considered is of order 10 in both order and degree and the Taylor method used is of order 10. Computations have been carried out in an Intel Xeon E5645 (2.40Ghz, 12Mb Cache and 5.86GT/s).

where we show the number of steps required by the different orders of the Taylor method. An order 5 Taylor method requires 145 steps while an order 15 Taylor method only requires 5. Therefore, although the order 5 requires less algebra at each step, similar algebra is repeated 29 times more and the final CPU time is increased by a factor of 7. Observe that this fact has been also taken into account in Section 2.3 when we computed the optimal step size for Taylor methods of integration of ODE's.

If we change the order and degree of the Earth potential we do not increase the number of iterates needed to achieve a required accuracy. This happens because the magnitude of the vector field does not change when the higher order and degrees of the Earth potential are added to the equations. It will be essentially a Two body problem plus small variations. Therefore, the time step predicted is approximately of the same for the two models.

Let us address now the CPU time consumption for other different perturbations than the ones coming from the Earth potential. Table 4.4 shows the CPU time required to compute the effect of the different perturbations (the harmonics we include, atmospheric drag and 3rd body perturbations) for the Jet Taylor method (top) and RK-78 method (bottom). As it is expected, the higher the order and the degree of the Earth potential the longer the CPU time. On the other hand, the effect of third bodies and drag depends only on the length of the integration time and therefore it is expected to be constant for all the cases. Observe that this happens for the RK78 (in all the three cases) and using both integration methods for the low orders of the Earth potential. However, for a high order and degree using the Jet Taylor method, longer times are needed to compute the Lunisolar effects and the atmospheric drag. This fact is stressed looking at the number of steps required in the Jet Taylor method. When we use high orders and degrees, the procedure needs smaller steps. Therefore, for a certain time span more steps are needed and, at each one, we have to compute the effect of the perturbations with the consequence that more CPU time is required.

Figure 4.17 shows the CPU time required as a function of the number of samples that we want to propagate. The time for a RK integration is approximately the same for all the individuals. For the

Taylor\potential	(2,2)	(4,4)	(8,8)	(16,16)	# steps
5	1.41	2.69	6.88	21.52	145
10	0.22	0.47	1.33	3.98	12
15	0.20	0.42	1.15	3.58	5
20	0.34	0.59	1.63	5.04	4

Table 4.3: Total CPU time needed for a 2 TU propagation ($\approx$ 25 minutes) as a function of different orders of the Taylor method and orders and degrees of the Earth potential considered. Computations have been performed in an Intel Xeon E5645 (2.40Ghz, 12Mb Cache and 5.86GT/s).

model	Earth potential	Lunisolar	drag	expected final time	total
0200	10.197	5.856	2.66	18.713	18.433
0202	14.277	5.66	2.604	22.541	23.133
1010	248.876	11.888	4.028	264.792	269.073
0200	38.57	108.751	8.613	155.934	157.042
0202	57.436	107.246	7.72	172.402	173.307
1010	441.956	101.166	8.728	551.85	556.155

Table 4.4: CPU time (in seconds) required for the propagation of 1000 initial conditions sampled with deviations in positions and velocities. The propagation time is of $T = 22$ hours and different perturbations (in columns) are included in the dynamical model. The upper block corresponds to the jet transport propagation and the lower one to the RK-78. Computations have been performed in an Intel Xeon E5645 (2.40Ghz, 12Mb Cache and 5.86GT/s).

Jet Taylor integration represents almost the same amount of work to compute one integration than 1000, assuming that all the initial conditions are close enough. Therefore, in Jet Taylor propagations, the time is almost constant independently of the number of initial conditions propagated. Also according to this plot, if we would perform a Monte Carlo sampling with few individuals (less than 600 in our case) it would be more efficient to use a RK-78. However, as a general rule samplings are usually much more crowded and the Jet Taylor method would be more advantageous in terms of CPU time.

4.2 Monte Carlo studies

The jet transport is very suitable to measure collision risk between satellites or between catalogued space debris and satellites. From the jet transport point of view, uncertainty regions are represented as boxes. Once the initial uncertainty box is propagated, we can determine the probability distribution of the final states of the satellites as a function of the initial distribution. In particular, if we consider the jet propagation of two different satellites, we can run a fast Monte Carlo simulation to compute statistics about the distance of a close passage.

Let us consider a case example consisting of two satellites in circular orbits that are going to collide after a given time T_c. Assume that their initial states are given by X_1 and X_2 and, in addition, let us consider that both orbits have a similar period so they are going to have other close approaches before collision at a certain times T_i (let us include in this T_i set the value of i such that $T_c = T_i$).

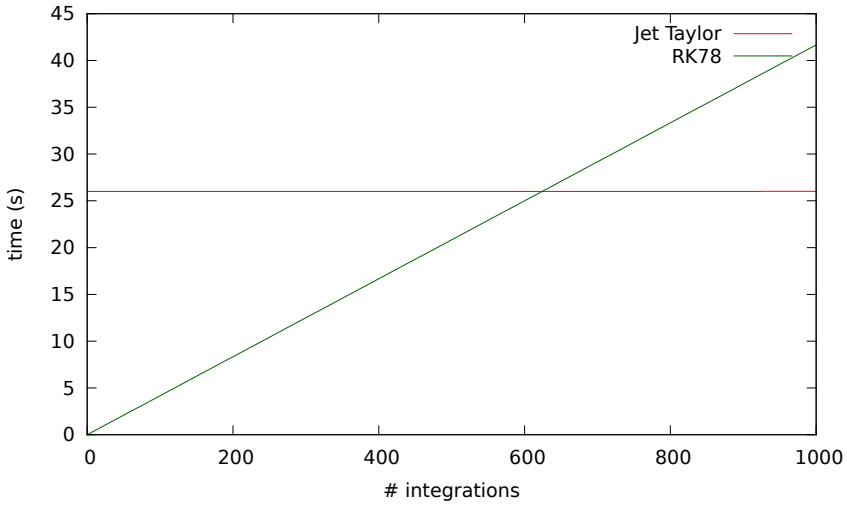


Figure 4.17: For an almost circular orbit around the Earth, CPU time (in seconds), needed for the propagations as a function of the number of initial conditions. Red: Jet Transport Taylor method of order 20. Green: RK-78 single state trajectory integrations. The computations have been performed with an Intel Xeon E5645 (2.40Ghz, 12Mb Cache, 5.86GT/s)

The test scenario is similar to the Iridium-Kosmos collision occurred on 2009. The satellites considered have similar orbital elements to ones of Iridium 33 and Kosmos-2251.

In Figure 4.18 we represent the initial position of the two satellites as well as their orbits around the Earth. The plot on the right shows a representation of the uncertainty ellipsoids for both spacecraft at the time of collision. Observe that they intersect in a small region when compared to the ellipsoid size.

In order to ensure the collision the two satellites are located at the same point with different velocities. Then the initial conditions of the problem are obtained propagating backwards in time for three revolutions. The period of the satellites is around 1.75 a-dimensional time units. Then we use the following procedure to study the close approaches and, in particular, the collision analysis.

1. Using the jet transport, propagate the neighbourhoods of both satellites to a time T_i. This means to compute the polynomial $P_1(\vec{\xi})$ (resp. $P_2(\vec{\eta})$) that gives us the state of the first (resp. second) satellite at time T_i when $\vec{\xi} = 0$. Other values of $\vec{\xi}$ (resp. $\vec{\eta}$) of the jet provide us the state of the corresponding satellite at time T_i for initial conditions $X_1 + \vec{\xi}$ (resp. $X_2 + \vec{\eta}$).

2. Take random initial conditions for the variables $\vec{\xi}$ and $\vec{\eta}$ following the probability distribution assumed in the initial uncertainty region (for instance a uniform or a Gaussian distribution).

3. Compute the distance of the two propagated satellites trajectories at time T_i. This means to compute the distance in the configuration space: $\|P_1(\vec{\xi})]_{(1,2,3)} - P_2(\vec{\eta})]_{(1,2,3)}\|_2$. Where $]_{(1,2,3)}$ means the first, second and third coordinates of the vectors, i.e. the configuration space variables.

4. Repeat steps 2 and 3 to obtain a representative statistical sampling (note that since the propagation is obtained as the result of an evaluation of a polynomial, each sample computation is very fast).

As a further remark we note that in step 3, when computing the distance at the final time between satellites corresponding to selected values of $\vec{\xi}$ and $\vec{\eta}$ in the initial uncertainty regions, the propagation time plays a role. For a fixed value of $\vec{\xi}$ and $\vec{\eta}$, there exist a time $\tau_i(\vec{\xi}, \vec{\eta})$ near T_i where the propagated states are closer than at time T_i, when the jets ($P_1(\vec{\xi})$ and $P_2(\vec{\eta})$) have been evaluated. To address this issue we introduce a time coordinate in the final box, also in a polynomial way, as we did in the computation of

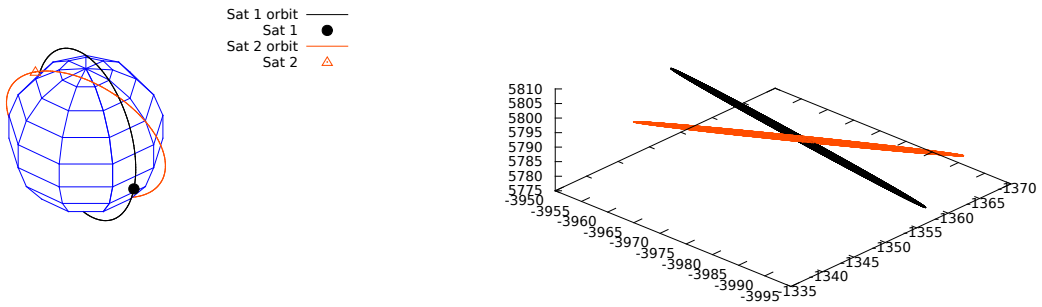


Figure 4.18: Left: Left: Initial conditions and orbits of the two spacecraft for the collision case study under consideration. The initial conditions are obtained integrating backwards in time the collision conditions to ensure a collision. Right: Uncertainty ellipsoids for both satellites at the time of collision.

Poincaré maps in chapter 2. This particular time $\tau_i(\vec{\xi}, \vec{\eta})$ where the minimum distance is attained, can be computed using a simple Newton method.

With the sampling results obtained we can compute some statistic indicators like its first moments (mean, standard deviation, skewness and kurtosis). These are reflected in the table of Figure 4.19 corresponding to different close approaches. Additionally, in the same figure we also show frequency histograms of the minimum distance together with the normal distribution obtained using the mean and the standard deviation resulting from the sampling.

As we can see in the figure, three of the four approaches have a mean distance of more than 40 kilometres while for the remaining one, corresponding to the third close encounter, the mean distance is about 500 meters. This third approach is the one corresponding to the real impact and we note that its mean is not zero since the minimum distance for each sample cannot be negative. Analysing the second indicator, the standard deviation, the longer the integration time the larger the standard deviation we find. This fact has also intuitive sense since for longer time integrations the uncertainty region increases and therefore there are more values in the uncertainty zone far away from the centre of the propagation.

The skewness, or third moment, seems to be the more relevant statistic indicator. In the colliding case it is high, while in the other ones it is close to zero. Intuitively this indicator is associated to the symmetry of the distribution. Positive values indicate that the samples are displaced to the left with respect to the normal distribution while negative ones indicate that they are displaced to the right. A zero value indicates that the samples follow a symmetric distribution. In our case, the non colliding encounters have very small skewness values, pointing to symmetric distributions. However, in the colliding scenario we have a large skewness indicating that there are more samples in the left part of the distribution. This lack of symmetry indication is reinforced by the fact that in a close encounter the mean distance of the distribution is small and we cannot have samples with negative distances in the histogram.

For the last indicator considered, the kurtosis or fourth moment, we have to say that it usually doesn't give any further information and similar values are found in all the approaches.

To roughly summarise the results of Figure 4.19, we mainly note that the normal distribution agrees quite well with the histogram for the non-colliding scenarios. On the other hand, for the colliding case it appears unbalanced to the left and this fact agrees with the skewness results.

4.3 Conclusions and future work

In the previous section we used a basic functionality of the Jet Transport for the analysis of satellite close approaches. We used the Jet to propagate the probability density function point-wise, in the sense that we

Approach	mean	standard deviation	skewness	kurtosis
First	49041.814935	92.760486	0.010182	-0.119546
Second	47710.973011	141.445321	0.034857	-0.133981
Third	545.739802	359.16694	0.634560	-0.128865
Fourth	44475.298463	376.657252	0.045448	-0.120202

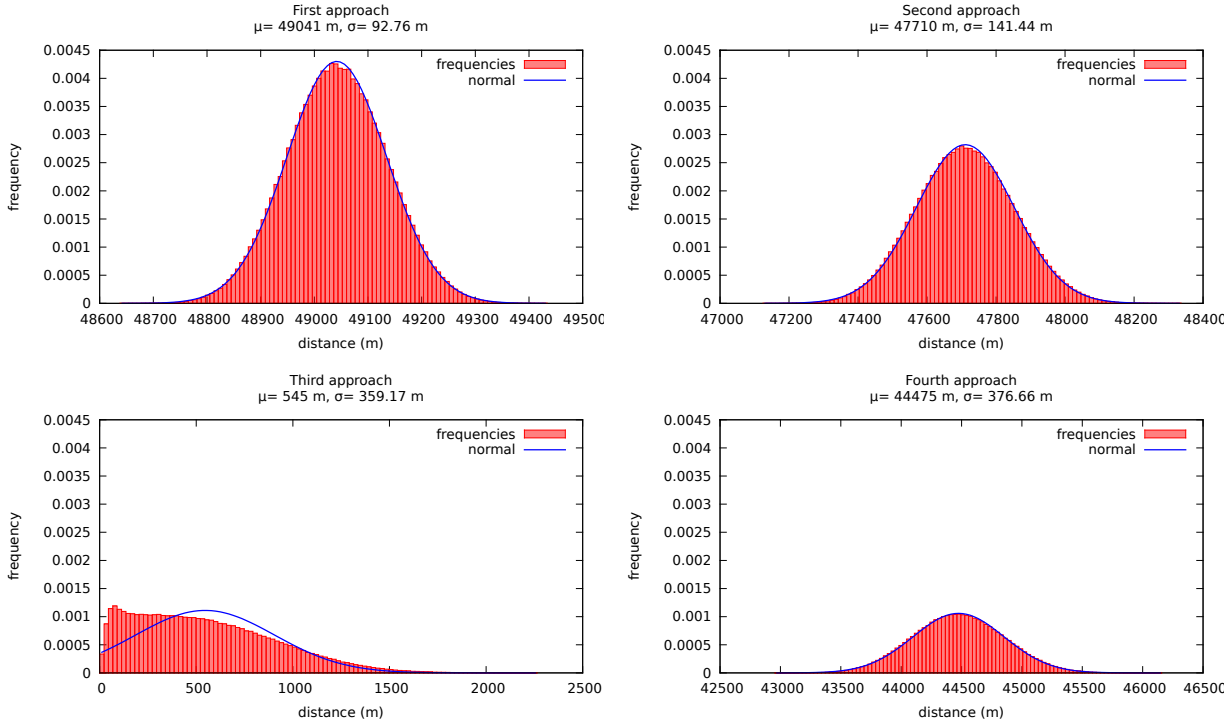


Figure 4.19: **Top:** The table shows some statistic indicators for the case example corresponding to collision risk of two satellites and four close approaches. **Bottom:** Frequency histograms of distances for the samples. From left to right and top to bottom, the first, second, third and fourth encounters are represented. As a reference, the blue line shows a normal distribution with the same mean and standard deviation as the corresponding samples.

sample using a given probability density function and then we use the Jet to propagate and to evaluate the indicators of each sample by means of polynomials. However, the Jet Transport offers us additional tools to study this problem. In chapter 2 we have seen that it is possible to propagate probability density functions using the Jet. Once we know the final PDF it is possible to know the probability for the spacecraft to be in a given region $\mathcal{U}$, just by integrating the polynomial on the region.

Let us assume that we have two spacecrafts in orbit around the Earth. The first one has a well known orbit. The second one is out of control and we only assume a PDF (f) associated to its position and velocity at an initial time T_0. Our objective is to evaluate the collision risk at a given time T of close approach. A general procedure would be as follows:

1. Propagate the uncontrolled spacecraft up to the time T using the Jet Transport procedure.
2. Compute the polynomial that approximates the PDF associated to the final state using the procedure of propagation of densities introduced in section 2.7.
3. Fix a neighbourhood $\mathcal{U}$ around the position of the first spacecraft at time T. For instance considering points with a distance to the spacecraft below a certain radius. Observe that this neighbourhood is taken in the configuration space and the collision may happen at any velocity (this causes some troubles that are discussed below).
4. Compute the collision probability by integrating the PDF in the region $\mathcal{U}$.

The main difficulty of the previous algorithm resides in the last step. The probability density function obtained after step 3 is on a six dimensional space while the neighbourhood $\mathcal{U}$ is in the configuration space which is 3-dimensional. Therefore the integration is done on $\mathcal{U}$ in configuration variables and on all the real numbers for the velocity variables, i.e. the velocity variables are integrated in the interval $(-\infty, \infty)$.

Unfortunately the polynomial approximation of the PDF obtained in step 2 approximates the PDF in a neighbourhood of the central position but never till infinity. Therefore, we may not be able to compute the exact probability, unless the PDF decays quickly to zero. In this case we may have good enough approximations of the collision probability integrating the PDF on the set of points where we estimate that is correctly computed.

When the position of both spacecraft are not well determined an additional step is required since the two spacecraft have to be propagated using the jet transport. Then the PDF f_1 (respectively f_2) associated to the probability that the first spacecraft, with coordinates $\vec{x}$, (respectively the second, with coordinates $\vec{y}$) be in the set $\mathcal{U}_1 \times \mathbb{R}^3$ (respectively $\mathcal{U}_2 \times \mathbb{R}^3$) at the time T_c must be approximated using similar polynomials as the ones computed in the preceding step. Then the velocity component must be reduced in order to have the PDF $f_1|_{\mathcal{U}_1}$ and $f_2|_{\mathcal{U}_2}$ associated to the event that the first, and the second spacecraft be in the region U_1 and U_2 respectively. As before this is done by integrating with respect to the velocity variables in the interval $(-\infty, \infty)$.

Next, the PDF of the distance between the two spacecrafts may be computed. Using the fact that $f_1|_{\mathcal{U}_1}$ and $f_2|_{\mathcal{U}_2}$ describe independent random variables, we can get the PDF, g, associated to the joint random variable describing the probability of the first spacecraft being in U_1 and the second one in U_2 by multiplying them, $g = f_1 f_2$. Finally to get the probability distance distribution we introduce the change of variables:

$$\begin{cases} z_0 = \|x - y\| & \\ z_i = x_1 & i = 1, 2 \\ z_{i+3} = y_i & i = 1, 2, 3. \end{cases}$$

To finish the procedure, the variables we are not interested in, i.e. z_i for $i > 0$, must be integrated in the interval $(-\infty, \infty)$ to obtain the PDF reduced to only distances.

Observe that our initial problem has 12 variables, six for each of the spacecraft. At the end of the procedure we only have one variable, the distance. Therefore, during the procedure we have reduced 11 variables and each of them has been reduced by integrating over all the real line, and as discussed before, it is not possible using local polynomial approximations. We are computing an approximation of it.

Another interesting problem that can be analysed in this context is the determination of parameter of the model. For instance, let us assume that there is a debris orbiting the Earth at a certain high, we know the position but we do not know its ballistic coefficient. Because of this, the predictions done for the object are not reliable. In order to improve them, it is possible to add additional variables in the jet to control deviations on the ballistic coefficient, and therefore, to obtain the final position as a function not only of the phase space variables, but also the ballistic coefficient. Then integrating debris up to a final time T we get a set of possible positions as a function of the ballistic coefficient deviation. If we perform a new observation of the same debris we can determine which of the values ballistic coefficient determination agrees with the new position and reduce this way the possible values of the ballistic coefficient. Iterating enough times the procedure we could reduce ballistic coefficient deviation to a suitable small range.

Appendix A

Software details

During this thesis several libraries has been developed to implement the algorithms. This appendix aims to collect and list the functions present in those libraries.

The first section of the appendix gives a list of the libraries developed. The main functions of each library are described.

The second section contains some implementing notes on parallelising using openMP. These come from an study of the parallelisation procedure done while implementing the division/selection procedure to detect LCS.

A.1 List of routines

The implementation of the algorithm developed in this thesis is classified in several libraries. The libraries developed are:

- `apfb.c`: basic functions of polynomial algebra (algebra polinomial funcions bàsiques in Catalan). This library contains the functions to initialise the polynomial algebra.
- `poliman.c`: polynomial manipulator. This library contains the functions to operate with polynomials.
- `taylor_jet.c`: This library contains the adapted Jet-Taylor method.
- `derivades.c`: it only contains the vector field written using the functions of the polynomial manipulator library. Should be written for each vector field.
- `poincare.c`: this library contains the implementation of the Poincaré map algorithms.
- `splitneighbours.c`: This library contains the implementation of the different algorithms for the splitting of neighbourhoods during the integration.
- `localinversions.c`: This library contains the implementation of the functional inversion algorithms.
- `densityprop.c`: This library contains the algorithms to propagate the density given an initial mapping.

The description of the functions is given in the following. For each library first the global variables introduced by the library commented, if it contains some. Then the functions are given. The prototype, the inputs and the o utputs together with a small description is given for each function.

apfb.c

The file apfb.c contains the necessary functions to work with a polynomial arithmetic. Here we can find functions to create and close the work-space, allocate and free polynomials as well as functions to change the notations between the multi-indices. The global variables introduced by this library are:

- `mdeg`: maximum degree of the polynomials in the work-space.
- `mvar`: maximum number of variables of the polynomials in the work-space.
- `mexp`: maximum exponent that can have a variable. It is limited by the number of variables *mvar* (is the maximum degree that the work-space can use).
- `**psi`: value of $\psi_v ar(deg)$. Each coefficient contains the number of monomials of degree *deg* with *var* variables.
- `**cmi`: `cmi[deg][cod]` contains the base 2^q representation of the monomial with degree `deg` and cardinal representation `cod`.
- `**divivar`: `divivar[deg][cod]` is factor for which one should multiply the coefficient of degree `deg` and cardinal representation `cod` to obtain the respective variational element. It is computed according to formula 2.9.

The functions implemented in this library are:

- `int poliar_ini(int maxdeg, int maxvar)`:
  Set up the polynomial arithmetic work-space with polynomials of maximum degree `maxdeg` and `maxvar` variables. Initialise the global variables `mdeg`, `mvar`, `mexp`, `**psi`, `**cmi` and `**divivar`. Must be called before calling any other polynomial algebra function.
  input:
  - `maxdeg`: maximum degree for the polynomial algebra.
  - `maxvar`: number of variables for the polynomial algebra.

  output:
  - `return` 0 if $variab > 8$, 2 otherwise.
- `void poliar_clo(int variab);`:
  Frees the global variables allocated in *poliar_ini*.
  input:
  - `var`: number of variables allocated in the polynomial algebra.

  output:
  - `return` void.
- `double** genpol(int deg,int variab;)`:
  Initialises a polynomial (defined as double**) with `variab` variables and degree `deg`. All the coefficients are 0.
  input:
  - `deg`: degree of the polynomial.
  - `deg`: number of variables of the polynomial.

  output:
  - `return` polynomial initialised.

- `void freepol(int deg,double **pol);`:
  Frees the polynomial `pol` generated using the function `genpol`.
  input:
  - `deg`: degree of the polynomial.
  - `**pol`: polynomial to be freed.

  output:
  - `return`: void

- `int nextlmon(int deg, int variab, int *mk);`:
  Given the multi-index `mk`, give the next multi-index in lexicographical order inside the multi-indices of `variab` variables and degree `deg`.

  input:
  - `deg`: order of the multi-index.
  - `variab`: dimension of the multi-index
  - `*mk`: multi-index given in vector notation.

  output:
  - `*mk`: the next multi-index in the lexicographical order of the multi-index given as input.
  - `return`: 0 if the input multi-index `mk` is the last monomial of degree `deg` with `variab` variables. Otherwise, the first coefficient of the multi-index different from 0.

- `int codemulti(int deg, int *mk, int variab);`:
  Given a multi-index `mk` in vector notation, it returns the multi-index in cardinal notation.
  input:
  - `deg`: order of the polynomial.
  - `*mk`: multi-index in vector notation.
  - `variab`: number of coefficients of the multi-index.

  output:
  - `return`: cardinal notation of the multi-index `mk`.

- `void ucodemulti (int code, int deg, int variab, int *mk);`:
  Given the cardinal notation `code` of a multi-index in `variab` variables and degree `deg` returns the corresponding vector notation.
  input:
  - `code`: cardinal notation of the multi-index.
  - `deg`: order of the multi-index
  - `variab`: number of coefficients of the multi-index.

  output:
  - `*mk`: multi-index in vector notation.
  - `return`: void.

- `void ucodemultibase(int base, int deg, int variab, int *mk);`:
  given the base 2^q notation of a monomial of order `deg` in `variab` variables returns the vector notation.
  input:

- `base`: base 2^q notation of the multi-index.
- `deg`: order of the multi-index.
- `variab`: number of coefficients of the multi-index.

output:

- `*mk`: vector notation of the multi-index.
- `return`: void.

- `void printpoli(double **pol,int var,int deg);`
Print the coefficients of the polynomial `**pol`. The coefficients are ordered by degrees and by reverse lexicographical order. Each coefficient is printed in a new line together with the exponents associated to the coefficient.
input:
  - `**pol`: coefficients of the polynomial.
  - `var`: number of variables of the polynomial.
  - `deg`: degree of the polynomial.

  output:
  - `return`: void.
  - `screen`: coefficients of `pol`.

- `void printpoliplot(double **pol,int var,int deg);`
Prints the coefficient of the polynomial and the exponents ready to be plotted in a plotting software like gnuplot.
input:
  - `**pol`: coefficients of the polynomial.
  - `var`: number of variables of the polynomial.
  - `deg`: maximum degree that must be printed.

  output:
  - `return`: void.
  - `screen`: coefficients of `pol`.

- `void printpolimd(double ***pol,int dim,int var,int deg);` Print the coefficient of a vector of polynomials.
input:
  - `***pol`: coefficients of the polynomials given in a vector.
  - `dim`: dimension of the vector.
  - `var`: number of variables of the polynomials.
  - `deg`: maximum degree of the polynomials to be printed.

  output:
  - `return`: void.
  - `screen`: coefficients of the polynomials.

poliman.c

This file contains all those functions that allows us to work with the polynomial algebra.
There are two kinds of functions. The first group is the standard multivariate polynomial operations. The second group consists in the same functions as before but prepared to be inserted in a Jet-Taylor scheme, i.e. they do the operations for a given degree j assuming that the lower degrees are known for both, the operands and the result polynomials.

The first variables are inputs and the last ones are outputs.

- `void addpoli(double **a, double **b, double **c);`:
  Computes the sum of two polynomials: $c(\vec{\xi}) = a(\vec{\xi}) + b(\vec{\xi})$. If the second variable (`b`) is `NULL` then the polynomial `a` is added to `c`: $c(\vec{\xi}) = c(\vec{\xi}) + a(\vec{\xi})$.
  input:
  - `**a`: coefficients of the polynomial $a(\vec{\xi})$.
  - `**b`: coefficients of the polynomial $b(\vec{\xi})$, or `NULL`.

  output:
  - `**c`: coefficients of $c(\vec{\xi})$.
  - `return`:void.
- `void prodpoli (double **a, double **b, double **c);`:
  Computes the product of two polynomials: $c(\vec{\xi}) = a(\vec{\xi}) * b(\vec{\xi})$.
  input:
  - `**a`: coefficients of the polynomial $a(\vec{\xi})$.
  - `**b`: coefficients of the polynomial $b(\vec{\xi})$.

  output:
  - `**c` coefficients of $c(\vec{\xi})$.
  - `return`: void.
- `void prodescpoli (double **a, double esc, double **c);`:
  Computes the product of a polynomial times a scalar: $c(\vec{\xi}) = \lambda \cdot a(\vec{\xi})$.
  input:
  - `**a`: coefficients of the polynomial $a(\vec{\xi})$.
  - `esc`: value of the scalar λ.

  output:
  - `**c` coefficients of $c(\vec{\xi})$.
  - `return`: void.
- `void divipoli (double **a, double **b, double **c);`:
  Computes the division of a polynomial $a(\vec{\xi})$ by another polynomial $b(\vec{\xi})$: $c(\vec{\xi}) = \frac{a(\vec{\xi})}{b(\vec{\xi})}$.
  If $b(\vec{\xi})$ has null independent term then the program is stopped.
  input:
  - `**a`: coefficients of the polynomial $a(\vec{\xi})$.

- `**b`: coefficients of the polynomial $b(\vec{\xi})$.

output:

- `**c` coefficients of $c(\vec{\xi})$.
- `return`: void.

- `void powpoli (double **b, double` α`, double **a);`:
Computes the power of $b(\vec{\xi})$ to the scalar value α.
If $b(\vec{\xi})$ has null independent term then the program is stopped.
input:
  - `**b`: coefficients of the polynomial $b(\vec{\xi})$.
  - `esc`: value of the scalar α.

  output:
  - `**a` coefficients of $a(\vec{\xi})$.
  - `return`: void.

- `void sincospoli (doulbe **a, double **s, double **c);`:
Computes the sinus and the cosine of a polynomial $a(\vec{\xi})$: $\begin{cases} s(\vec{\xi}) = \sin(a(\vec{\xi})) \\ c(\vec{\xi}) = \cos(a(\vec{\xi})) \end{cases}$.
If only one of them is needed the other should be also included since both of them are necessary to complete the computation.
input:
  - `**a`: coefficients of the polynomial $a(\vec{\xi})$.

  output:
  - `**s`: coefficients of the polynomial $s(\vec{\xi})$.
  - `**c`: coefficients of the polynomial $c(\vec{\xi})$.
  - `return`: void.

- `void exppoli (double **b, double **a);`:
Compute the exponential of the polynomial $b(\vec{\xi})$, $a(\vec{\xi}) = \exp(b(\vec{\xi}))$.
input:
  - `**b`: coefficients of the polynomial $b(\vec{\xi})$.

  output:
  - `**a`: coefficients of the polynomial $a(\vec{\xi})$.
  - `return`: void.

- `void logpoli (double **b, double **a);`:
Computes the logarithm of the polynomial $b(\vec{\xi})$, $a(\vec{\xi}) = \log(b(\vec{\xi}))$
input:
  - `**b`: coefficients of the polynomial $b(\vec{\xi})$.

  output:

- `**a`: coefficients of the polynomial $a(\vec{\xi})$.
- `return`: void.

Functions related to differential polynomial algebra.

- `void derpol (double **pol, int var, double **dpol);`:

  Computes the derivative of the polynomial $p(\vec{\xi})$ with respect to the variable `var`, $dp(\vec{\xi}) = \frac{\partial p(\vec{\xi})}{\partial \vec{\xi}_{var}}$.

  input:

  - `**pol`: coefficients of the polynomial $p(\vec{\xi})$.
  - `var`: index of the variable with respect we differentiate.

  output:

  - `**dpol`: coefficients of the polynomial $dp(\vec{\xi})$.
  - `return`: void.

- `void difpol(double ***pol, int n, double ****dpol);`:
  Computes the differential matrix of a vector of polynomials $P(\vec{\xi})$, $D_{\vec{\xi}}P(\vec{\xi})$.
  input:

  - `***pol`: coefficients of the polynomial $P(\vec{\xi})$ given as a vector.
  - `n`: dimension of the vector of polynomials.

  output:

  - `***dpol`: differential matrix $D_{\vec{\xi}}P(\vec{\xi})$.
  - `return`: void.

- `void jacobianpoli(double ***pol, int n, double **jac);`:
  Computes the Jacobian of the vector polynomial $P(\vec{\xi})$, $|D_{\vec{\xi}}P(\vec{\xi})|$.
  input:

  - `***pol`: coefficients of the polynomial $P(\vec{\xi})$ given as a vector.
  - `n`: dimension of the vector of polynomials.

  output:

  - `**jac`: coefficients of the polynomial $|D_{\vec{\xi}}P(\vec{\xi})|$.
  - `return`: void.

- `void integrapoli(double **x, int var, double **intx);`:
  Computes the indefinite integral of the polynomial $P(\vec{\xi})$ with respect the variable `var`, $I(\vec{\xi}) = \int P(\vec{\xi}) d\vec{\xi}_i$
  input:

  - `**x`: coefficients of the polynomial $P(\vec{\xi})$.
  - `var`: integration variable, i.

  output:

  - `**intx`: coefficients of the polynomial $I(\vec{\xi})$.
  - `return`: void.

- `void integrapoli(double **x, int var, double xinf, double xsup, double **intx);`:
  Computes the definite integral of the polynomial $P(\vec{\xi})$ with respect the variable `var` between a and b, $I(\vec{\xi}) = \int_a^b P(\vec{\xi}) d\vec{\xi}_i$
  input:
  - `**x`: coefficients of the polynomial $P(\vec{\xi})$.
  - `xinf`: inferior bound of the integral, a.
  - `xsup`: superior bound of the integral, b.
  - `var`: integration variable, i.

  output:
  - `**intx`: coefficients of the polynomial $I(\vec{\xi})$.
  - `return`: void.

Functions related to the evaluation of polynomials.

- `double hornermv (double **poli,double *xval, int grm, int gra, int varact);`:
  Evaluation of the polynomial $P(\vec{\xi})$ at the point $\vec{\xi_e}$, $ev = P(\vec{\xi_e})$. The function is recursive, the variables `grm` and `varact` changes from one step of the recursion to the next one.
  input:
  - `**poli`: coefficients of the polynomial $P(\vec{\xi})$.
  - `*xval`: vector of variables that is evaluated, $\vec{\xi_e}$.
  - `grm`: degree of the polynomial.
  - `gra`: actual degree of the polynomial, when called at first time it contains the maximum degree, i.e. $grm = gra$. Then it is reduced according to the necessity in the recursion.
  - `varact`: number of variables of $P(\vec{\xi})$.

  output:
  - `return`: value of the evaluation, $P(\vec{\xi_e})$.

- `void hornermvpoli (double **poli,double ***xval, int grm, int gra, int varact, double **ret);`:
  Composition of polynomials. Given the polynomials $P(\vec{\xi})$ and $Q(\vec{\xi})$ it computes the polynomial $P(Q(\vec{\xi}))$. It is based in the evaluation of polynomials considering the the evaluation vector with polynomial coefficients. The function is recursive, the variables `grm` and `varact` changes from one step of the recursion to the next one.
  input:
  - `**poli`: coefficients of the polynomial $P(\vec{\xi})$.
  - `***xval`: vector of polynomial coefficients, $Q(\vec{\xi})$.
  - `grm`: degree of the polynomial.
  - `gra`: actual degree of the polynomial, when called at first time it contains the maximum degree, i.e. $grm = gra$. Then it is reduced according to the necessity in the recursion.
  - `varact`: number of variables of $P(\vec{\xi})$.

  output:
  - `**ret`: Result of the composition, $P(Q(\vec{\xi}))$.

- `return`: Void.

Functions to evaluate the maximum admissible values of the initial boxes.

- `double maxbox(double ***x, int graupoli, int variables);`:
  Given a polynomial $P(\vec{\xi})$, computes the maximum size that can be propagated with enough accuracy, $\vec{\xi}_{\max}$, according to formula 2.7
  input:
  - `***P`: coefficients of the polynomial $P(\vec{\xi})$.
  - `graupoli`: degree of the polynomial $P(\vec{\xi})$.
  - `variables`: number of variables of the polynomial $P(\vec{\xi})$.

  output:
  - `return`: initial maximum size to ensure an accuracy of 10^{-6}, $\vec{\xi}_{\max}$.

Functions of the polynomial linear algebra, they are based on the implementation of the standard LU decomposition implemented by M. Marcote.

- `void ludecpoli(double ****a,int n, int *indx, double *d,int oj,int var);`:
  Computes the LU decomposition of the matrix A with polynomial coefficients.
  input:
  - `****a`: coefficients of the polynomial components of the matrix A.
  - `n`: dimension of the matrix A.
  - `oj`: maximum degree of the polynomial components.
  - `var`: number of variables of the polynomial components.

  output:
  - `*indx`: permutation vector of LU decomposition.
  - `*d`: number of pivots in the LU decomposition.
  - `****a`: coefficients of the LU decomposition of the matrix A.
  - `return`: void.

- `void lubksbpoli(double ****a,int n, int oj, int var, int *indx,double ***b);`:
  Computes the backwards substitution resolution to solve the system $Ax = b$ with polynomial components in A and b. The solution x also contains polynomial components.
  input:
  - `****a`: coefficients of polynomials given by the LU decomposition.
  - `n`: dimension of system of equations.
  - `oj`: degree of the polynomials of the components of A.
  - `var`; number of variables of the polynomial components.
  - `*index`: permutation vector obtained in the LU decomposition.
  - `***b`: independent term of the system of equations, b.

  output:
  - `***b`: solution of the system of equations, x.
  - `return`: void.

The second group of functions in this file are those ones that are prepared to be directly integrated in a Taylor scheme. All of them are computed in such way that returns all the monomials of degree j in the polynomial given in the last variable. For more details of how to use them see how to write the field file. Observe that now the variables are arrays in 3 dimensions. The first dimension controls the powers of t while the second and third control the polynomial coefficient of the respective order. Therefore, this kind of variables are polynomial in t whose coefficients are polynomial in the states x.
note:
The addition and the product by an scalar have not a special version since the standard polynomial manipulation version can be used giving the j-th coordinate to the function:

- `addpoli(a[j],b[j],c[j])`
- `prodescpoli(a[j],esc,c[j])`

The remaining operations are:

- `void prodpoli_term (double ***a, double ***b, int j, double ***c);`:
  Computes the polynomial coefficient of degree j in t of the product $c(t,\vec{\xi})\Big|_{[j,\cdot]} = \left(a(t,\vec{\xi}) \cdot b(t,\vec{\xi})\right)\Big|_{[j,\cdot]}$.
  input:
  - `***a`: polynomial $a(t,\vec{\xi})$.
  - `***b`: polynomial $b(t,\vec{\xi})$.
  - `j`: exponent of the expansion in time variable j that we want to compute.

  output:
  - `***c`: polynomial $c(t,\vec{\xi})$.
  - `return`: void.
- `void divipoli_term (double ***a, double ***b, int j, double ***c);`:
  Computes the polynomial coefficient of degree j in t resulting of the division: $c(t,\vec{\xi})\Big|_{[j,\cdot]} = \left(\frac{a(t,\vec{\xi})}{b(t,\vec{\xi})}\right)\Big|_{[j,\cdot]}$.
  input:
  - `***a`: polynomial $a(t,\vec{\xi})$.
  - `***b`: polynomial $b(t,\vec{\xi})$.
  - `j`: exponent of the expansion in time variable j that we want to compute.

  output:
  - `***c`: polynomial $c(t,\vec{\xi})$.
  - `return`: void.
- `void powpoli_term(doulbe ***a, double alpha, int j, double ***c);`:
  Computes the polynomial coefficient of degree j in t resulting of power up the polynomial $a(\vec{\xi})$ to α: $c(t,\vec{\xi})\Big|_{[j,\cdot]} = a(\vec{\xi})^{\alpha}\Big|_{[j,\cdot]}$.
  input:
  - `***a`: polynomial $a(t,\vec{\xi})$.
  - `alpha`: scalar value, α, up to which the power is computed.
  - `j`: exponent of the expansion in time variable j that we want to compute.

output:

- `***c`: polynomial $c(t,\vec{\xi})$.
- `return`: void.

- `sincospoli_term(double ***a, int j, double ***s, double ***c);`:
  Computes the polynomial coefficient of degree j in t resulting to apply the sinus and the cosine (respectively s and c) to the polynomial $a(\vec{\xi})$: $\begin{cases} s(t,\vec{\xi})\Big|_{[j,\cdot]} = \sin(a(t,\vec{\xi}))\Big|_{[j,\cdot]} \\ c(t,\vec{\xi})\Big|_{[j,\cdot]} = \cos(a(t,\vec{\xi}))\Big|_{[j,\cdot]} \end{cases}$
  input:
  - `***a`: polynomial $a(t,\vec{\xi})$.
  - `j`: exponent of the expansion in time variable j that we want to compute.

  output:
  - `***s`: polynomial $s(t,\vec{\xi})$.
  - `***c`: polynomial $c(t,\vec{\xi})$.
  - `return`: void.

- `void exppoli_term (double ***b, int j, double ***a);`:
  Computes the polynomial coefficient of degree j in t resulting of computing the exponential of the polynomial $b(\vec{\xi})$: $a(t,\vec{\xi})\Big|_{[j,\cdot]} = \exp(b(\vec{\xi}))\Big|_{[j,\cdot]}$.
  input:
  - `***b`: polynomial $b(t,\vec{\xi})$.
  - `j`: exponent of the expansion in time variable j that we want to compute.

  output:
  - `***a`: polynomial $c(t,\vec{\xi})$.
  - `return`: void.

- `void logpoli_term (double ***b, int j, double ***a);`:
  Computes the polynomial coefficient of degree j in t resulting of computing the logarithm of the polynomial $b(\vec{\xi})$: $a(t,\vec{\xi})\Big|_{[j,\cdot]} = \log(b(\vec{\xi}))\Big|_{[j,\cdot]}$.
  input:
  - `***b`: polynomial $b(t,\vec{\xi})$.
  - `j`: exponent of the expansion in time variable j that we want to compute.

  output:
  - `***a`: polynomial $c(t,\vec{\xi})$.
  - `return`: void.

taylor_jet.c

This library contains the functions related to the flow propagation with Jets. It contains a function to obtain the state at time t starting at time t_0, as well as different implementations of step controls.

- `int taylorjet (doulbe *t, double ***x, int n, double *h, double hmin, double hmax, double tfinal, double tol, int p, int jet);`:
  Given an actual time t, the actual states $x_t(\vec{\xi})$ and an approximation of the step size h_a, this function computes a new step size h (between $h_{\min}$ and $h_{\max}$ and according to a given tolerance, see section 2.3), the states $x_{t+h}(\vec{\xi})$ at time $t+h$ and updates the time t. If the final time t_f is reached, the function returns the states at that final time. The differential equations integrated depends on the function `derivades` (see later).
  input:
  - `*t`: actual time.
  - `***x`: coefficients of the polynomial $x_t(\vec{\xi})$.
  - `n`: dimension of the system of differential equations.
  - `*h`: approximation of the step size, h_a.
  - `hmin`: minimum step size, $h_{\min}$.
  - `hmax`: maximum step size, $h_{\max}$.
  - `tfinal`: desired final time, t_f.
  - `tol`: tolerance that the new step must fulfil.
  - `p`: order of the Taylor expansion in time.
  - `jet`: degree of the state variable polynomials.

  output:
  - `*t`: time after the integration step, $t + h$.
  - `***x`: coefficients of the polynomial $x_{t+h}(\vec{\xi})$.
  - `*h`: size of the step size.
  - `return`: 0 if the step size is successful, 1 if the final time t_f is reached or -1 if there is some problem during the integration step.

- `int taynostp(double *t, double ***x, double ****dz, int n, double *h, double tol, int p,int jet);`:
  Given an actual time t and the actual states $x_t(\vec{\xi})$, this function computes the Taylor expansion in time and position of the flow map $\phi(t+\tau; t_0, x_0 + \vec{\xi})$. It also computes the maximum size, h, in time that can be computed according to the procedure developed in section 2.3.
  input:
  - `*t`: actual time.
  - `***x`: coefficients of the polynomial $x_t(\vec{\xi})$.
  - `n`: dimension of the system of differential equations.
  - `tol`: tolerance that the new step must fulfil.
  - `p`: order of the Taylor expansion in time.
  - `jet`: degree of the state variable polynomials.

  output:

- `****dz`: coefficients of the polynomial $x_t(\vec{\xi},\tau)$ approximating the flow map $\phi(t+\tau;t_0,x_0+\vec{\xi})$.
- `*h`: approximation of the step size, h_a.
- `return`: 0.

- `double stctlr1(double ****dz, double tol,double hmi, double hma, int n ,int p, int jet);`:
  Step control strategy, it uses the last terms in the time expansion and the order 0 in the state variable expansion. It returns the step control of a non Jet implementation.
  input:
  - `****b`: coefficients of the Taylor expansion of the flow map $\phi(t+\tau;t_0,x_0+\vec{\xi})$ in time τ and phase space variables $\vec{\xi}$.
  - `tol`: tolerance that must satisfy the step size.
  - `hmi`: minimum step size acceptable.
  - `hma`: minimum step size acceptable.
  - `n`: dimension of the system.
  - `p`: order of the Taylor expansion in time.
  - `jet`: degree of the polynomials in the Jet phase space variables.

  output:
  - `return`: step size obtained h.

- `double stctlr1(double ****dz, double tol,double hmi, double hma, int n ,int p, int jet);`:
  Step control strategy, it uses the last terms in the time expansion and the all the orders in the state variable expansion. This function uses all the orders in the phase space expansion to approximate the step size.
  input:
  - `****b`: coefficients of the Taylor expansion of the flow map $\phi(t+\tau;t_0,x_0+\vec{\xi})$ in time τ and phase space variables $\vec{\xi}$.
  - `tol`: tolerance that must satisfy the step size.
  - `hmi`: minimum step size acceptable.
  - `hma`: minimum step size acceptable.
  - `n`: dimension of the system.
  - `p`: order of the Taylor expansion in time.
  - `jet`: degree of the polynomials in the Jet phase space variables.

  output:
  - `return`: step size obtained h.

field.c

The file field.c computes the normalised derivatives of a field to be used in the `taylorjet` function. It contains the information of the vector field used in the unique function `derivades`. Each vector field is implemented in a different file.

- `int derivades(double *t, double ***x, int n, double ****dz, int p,int jet);`:
  Given an initial condition $x_t(\vec{\xi})$ at time t, this function computes the normalised derivatives dz of a given vector field, i.e. the coefficients (polynomials of degree *jet* in the states $\vec{\xi}$) of the Taylor expansion up to order p in time of the solution around the initial condition.
  input:

- `*t`: actual time, t.
- `***x`: coefficients of the polynomial $x_t(\vec{\xi})$.
- `n`: dimension of the system of differential equations.
- `p`: order of the Taylor expansion in time.
- `jet`: degree of the state variable polynomials.

output:

- `****dz`: coefficients of the polynomial $x_t(\vec{\xi}, \tau)$ approximating the flow map $\phi(t+\tau; t_0, x_0 + \vec{\xi})$.
- `return`: 0 if there are not errors, 1 otherwise.

The following vector-fields has been implemented:

- CR3BP: see equations 1.2.
- Earth perturbation vector field: see chapter 4.
- ER3BP: see equations 1.3.
- The fish vector field: $\ddot{x} = -x + x^2$.
- Kepler problem vector field.
- Linear saddle $\dot{x} = x, \quad \dot{y} = -y$.
- Linear centre $\dot{x} = y, \quad \dot{y} = -x$.
- Linear node $\dot{x} = x, \quad \dot{y} = 2x$.
- Pendulum, $\ddot{x} = -\sin x$.
- Perturbed pendulum $\ddot{x} = (2.5\cos 5t - 1)\sin(x)$.

An specially long vector field to code is the vector field of the Earth perturbations. In particular, the computation of the full Earth perturbed potential. Several auxiliary variables has been defined in a vector u. Table A.1 shows the relation between the name of the auxiliary variable and the position in the vector u.

We will distinguish two different types of variables. The first ones are independent variables that are all of them grouped in the beginning of the vector `u`. The second group are the auxiliary variables related with V and W coefficients in the Earth potential vector field. To classify this second group we need the first vector of indices `rvn` which measures the first index of different variables. Then the element of the matrix is accessed using the vector lm which give the initial vector for triangular matrices. Finally the last index is given just by addition. Summarising, to access $V*_{i,j}$ or $W*_{i,j}$ the vector `u` is accesses by calling: `u[ rvn[var] + lm[i] + j]`.

poincare.c

This file contains the functions to compute the Poincaré map of a vector field.

- `double poincare map (double t, double ***x, int n, int jet, int meth);`:
  Computes the Poincaré map of the initial point $x_{t_0}(\vec{\xi})$. It uses the `taylorjet_poincare` to propagate the function to the Poincaré section $\mathcal{M}$.
  input:
  - `t`: initial time of the integration.
  - `***x`: coefficients of the polynomial $x_{t_0}(\vec{\xi})$.

variable	component of u	calling vector
x	1	–
y	2	–
z	3	–
x^2	3	–
y^2	4	–
z^2	5	–
r^2	6	–
$u_{aux,3} = r$	7	–
$\rho = \frac{R_\oplus}{r^2}$	8	–
x_n	9	–
y_n	10	–
z_n	11	–
$u_{aux,4} = acc_{aux}$	12	–
$u_{aux,5} = acc_{aux}$	13	–
$u_{aux,6} = acc_{aux}$	14	–
$V[i][j]$	rvn[0]	lm[i]+j
$W[i][j]$	rvn[1]	lm[i]+j
$Vx[i][j]$	rvn[2]	i
$Wx[i][j]$	rvn[3]	i
$Vy[i][j]$	rvn[4]	i
$Wy[i][j]$	rvn[5]	i
$Vz[i][j]$	rvn[6]	lm[i-1]+j
$Wz[i][j]$	rvn[7]	lm[i-1]+j
$Vr[i][j]$	rvn[8]	lm[i]+j
$Wr[i][j]$	rvn[9]	lm[i]+j

Table A.1: Variables involved in the perturbed Kepler for the oblate planet.

- `n`: dimension of the system used.
- `jet`: degree of the polynomials in the $\vec{\xi}$ variable.
- `meth`: method used to compute the Poincaré map, see section 2.4 for more details:
  - 1: Euler method.
  - 2: General iterative procedure.
  - 3: Newton method.

output:

- `***x`: coefficients of the polynomial Poincaré map $P_{\mathcal{M}}(x_0 + \vec{\xi})$.
- `return`: final time of intersection with the Poincaré section $\mathcal{M}$ for the point x_0.

- `int taylorjet_poincare(double *t, double ***x, int n, double *h, double hmi, double hma, double tfinal, double tol, int p,int jet,double vy0);`:
  Computes an integration step with the corresponding vector field with for the state $x_t\vec{\xi}$ at time t like the function `taylorjet` in the `taylor_jet.c` library. If the propagation of the flow crosses the Poincaré's section $\mathcal{M}$ it returns the polynomial $x_t(\vec{\xi})$ such that $x_t(0) \in \mathcal{M}$.
  input:
  - `*t`: actual time.
  - `***x`: coefficients of the polynomial $x_t(\vec{\xi})$.
  - `n`: dimension of the system of differential equations.
  - `*h`: approximation of the step size, h_a.
  - `hmin`: minimum step size, $h_{\min}$.
  - `hmax`: maximum step size, $h_{\max}$.
  - `tfinal`: desired final time, t_f.
  - `tol`: tolerance that the new step must fulfil.
  - `p`: order of the Taylor expansion in time.
  - `jet`: degree of the state variable polynomials.
  - `vy0`: constant of definition of the Poincaré's section $\mathcal{M} = \{x \in \mathbb{R}^n | x]_n = vy0\}$.

  output:
  - `*t`: time after the integration step, $t + h$.
  - `***x`: coefficients of the polynomial $x_{t+h}(\vec{\xi})$.
  - `*h`: size of the step size.
  - `return`: 0 if the step size is successful, 1 if the final time t_f is reached, 2 if the Poincaré's section is reached or -1 if there is some problem during the integration step.

- `void hiorpoindt (double ***dz,int op, int var, int jet, double ***dt);`:
  High order Poincaré (map) computation with a general iterative procedure to determine the function $\tau(\vec{\xi})$, considering the general Poincaré section $\mathcal{M} = \{x \in \mathbb{R}^n | G(x) = 0\}$.
  input:
  - `***dz`: coefficients of the polynomial $G(x_t(\vec{\xi}, \tau))$, $x_t(\vec{\xi}, \tau)$ must be such that $x_t(0, 0) \in \mathcal{M}$.
  - `op`: order of the Taylor expansion in time.
  - `var`: number of variables in the system.
  - `jet`: degree of the state variable polynomials.

output:

- `***dt`: `**dt[i]` contains the coefficients of the polynomial $\tau^i(\vec{\xi})$.
- `return`: void.

- `void hiornewpoindt (double ****dz,int op, int var, int jet, double ***pot);`:
  High order Poincaré (map) computation with a Newton's method to determine the function $\tau(\vec{\xi})$, considering the general Poincaré section $\mathcal{M} = \{x \in \mathbb{R}^n | G(x) = x]_n = 0\}$.
  input:
  - `****dz`: coefficients of the polynomials approximating $\phi(t+\tau; t_0, x_0+\vec{\xi})$, such that $\phi(t+0; t_0, x_0+0) \in \mathcal{M}$.
  - `op`: order of the Taylor expansion in time.
  - `var`: number of variables in the system.
  - `jet`: degree of the state variable polynomials.

  output:
  - `***pot`: `**dt[i]` contains the coefficients of the polynomial $\tau^i(\vec{\xi})$.
  - `return`: void.

spltitneighbours.c

This library contains all the functions related to the splitting of neighbourhoods during the integration procedure..

The first functions are related to the splitting polynomials algorithm.

- `int integrasplit (double ***x, double t0, double tf, double **info,double ****pfin,int n,int oj);` :
  Given the initial state $x_{t_0}(\vec{\xi})$ at time t_0 it propagates the state up to the final time t_f, if at some point the accuracy of the polynomials is not good enough, then the polynomial is splitted using `divideix`. The information is stored in `**info` and `****tfin`.
  input:
  - `***x`: coefficients of the initial state polynomial $x_{t_0}(\vec{\xi})$.
  - `t0`: initial time t_0.
  - `tf`: final time t_f.
  - `n`: number of variables in the system.
  - `oj`: degree of the state variable polynomials.

  output:
  - `**info`: information of the centre of the polynomials and the directions in which each splitting polynomial is produced.
  - `****pfin`: coefficients of the polynomials at the final state.
  - `return`: number of polynomials splitted up to that moment, is the effective length of `**info`.

- `void divideix (int i, double **poli,double **nou,int grau, int var);` :
  Given a polynomial $x_t(\vec{\xi})$ and a direction i this function generates two new polynomials $x_t(\vec{\xi_1}, \ldots, \vec{\xi_i}/2 \pm 1/2, \ldots, \vec{\xi_n})$.
  input:

- `i`: Direction in which the polynomial must be splitted.
- `**poli`: coefficients of the polynomial that must be splitted.
- `grau`: degree of the state variable polynomials.
- `var`: number of variables in the system.

output:

- `**poli`: coefficients of the polynomial after the splitting procedure, $x_t(\vec{\xi}_1, \ldots, \vec{\xi}_i/2 + 1/2, \ldots, \vec{\xi}_n)$.
- `**nou`: coefficients of the polynomial after the splitting procedure, $x_t(\vec{\xi}_1, \ldots, \vec{\xi}_i/2 - 1/2, \ldots, \vec{\xi}_n)$.
- `return`: void.

The second group of functions is related to the splitting of neighbourhoods.

- `int inteidivi (double *xini, double t0, double tf, double **info, double ****pfin, int n, int oj);`:
  Given the initial state $x_{t_0}(\vec{\xi})$ at time t_0 it propagates the state up to the final time t_f, if at some point the accuracy of the polynomials is not good enough, then the polynomial is splitted using `divideix`. The information is stored in `**info` and `****pfin`.
  input:
  - `*xini`: initial conditions around which we want to propagate neighbourhoods.
  - `t0`: initial time t_0.
  - `tf`: final time t_f.
  - `*info[0]`: information required for the first neighbourhood, its centre and radius that must be propagated.
  - `n`: number of variables in the system.
  - `oj`: degree of the state variable polynomials.

  output:
  - `**info`: information of the centres and radii of the neighbourhoods that must be propagated at each step.
  - `****pfin`: coefficients of the polynomials at each step of the procedure.
  - `return`: if the final time is reached returns the number of neighbourhoods computed, otherwise returns -1.

- `int novescaixesrec (double ****pfin, double **info, int *inbo, int *ndiv, double t, int n,int oj);`:
  This function adds the necessary new neighbourhoods following the divide and conquer algorithm for the neighbourhood division. It receives the polynomials $P_i(\vec{\xi})$ which propagates the initial points, then the maximum distance is search to determine the direction of division, the radius of the neighbourhood is fixed to be minimum and if two centres are two close both neighbourhoods are joined.
  input:
  - `****pfin`: coefficients of the polynomials at each step of the procedure.
  - `**info`: information of the centres and radii of the neighbourhoods that must be propagated at each step.
  - `*inbo`: index of the first neighbourhood propagated in the last integration step.
  - `*ndiv`: number of neighbourhoods propagated in the last integration step.
  - `n`: dimension of the vector field.

– `oj`: degree of the state variable polynomials.

output:

– `**info`: information of the new boxes, centres and radii.

– `*inbo`: index of the first neighbourhood generated by the present calling of the function.

– `*ndiv`: number of divisions done in the present calling of the function.

– `return`: 1 if everything runs fine, -1 if the maximum number of boxes prefixed is reached.

- `int novcapunts (double **puntrec, int npu,double ****pfin, double **info, int *inbo, int *ndiv, double t, int n,int oj);` :
Implementation of the algorithm 1 of Section 2.5. Given the polynomials $P_i(\vec{\xi})$ obtained in the last integration step, and the set of points that acts as tracers the new neighbourhoods are defined.
input:

  – `**puntrec`: set of points that must be covered by the new neighbourhoods, $\vec{t}_{n,i}$.

  – `npu`: number of points in the set of points that must be covered.

  – `****pfin`: coefficients of the polynomials obtained in the previous steps of the algorithm.

  – `**info`: information of the centres and radii of the previous neighbourhoods propagated.

  – `*inbo`: index of the first neighbourhood propagated in the last integration step.

  – `*ndiv`: number of neighbourhoods propagated in the last integration step.

  – `t`: time of integration.

  – `n`: dimension of the vector field.

  – `oj`: degree of the state variable polynomials.

  output:

  – `**info`: information of the new boxes: centres and radii obtained after applying the algorithm.

  – `*inbo`: number of neighbourhoods propagated in total.

  – `*ndiv`: number of divisions performed in the last division step.

  – `return`: 0.

- `int novcapuntsnorm (double **puntrecini, int npu,double ****pfin, double **info, int *inbo, int *ndiv, double t, int n,int oj);` :
Implementation of the algorithm 2 of Section 2.5. Given the polynomials $P_i(\vec{\xi})$ obtained in the last integration step, and the set of points that acts as tracers the new neighbourhoods are defined.
input:

  – `**puntrecini`: coordinates of the tracers that must be covered by the new neighbourhoods, $\vec{t}_{n,i}$.

  – `npu`: number of points in the set of points that must be covered.

  – `****pfin`: coefficients of the polynomials obtained in the previous steps of the algorithm.

  – `**info`: information of the centres and radii of the previous neighbourhoods propagated.

  – `*inbo`: index of the first neighbourhood propagated in the last integration step.

  – `*ndiv`: number of neighbourhoods propagated in the last integration step.

  – `t`: time of integration.

  – `n`: dimension of the vector field.

  – `oj`: degree of the state variable polynomials.

output:

- `**info`: information of the new boxes: centres and radii obtained after applying the algorithm.
- `*inbo`: number of neighbourhoods propagated in total.
- `*ndiv`: number of divisions performed in the last division step.
- `return`: 0.

- `int propsonda (double **sonda, int npu, double ****pfin, double **info, int inbo, int n, int oj); :`
Propagates the tracers $t_{n,i}$ to the next position $t_{n+1,i}$. If it is necessary adds a new tracer calling `addsonda`.[o]
input:
  - `**sonda`: coordinates of the tracers that must be covered by the new neighbourhoods, $\vec{t}_{n,i}$.
  - `npu`: number of tracers.
  - `****pfin`: coefficients of the polynomials obtained in the previous steps of the algorithm.
  - `**info`: information of the centres and radii of the previous neighbourhoods propagated.
  - `inbo`: index of the first neighbourhood propagated in the last integration step.
  - `n`: dimension of the vector field.
  - `oj`: degree of the state variable polynomials.

  output:
  - `**sonda`: coordinates of the tracers for the next step, $\vec{t}_{n+1,i}$.
  - `return`: number of tracers at the end of the procedure.

- `int addsonda (double **sonda, int npu, int index, double ****pfin, double **info, int inbo, int n, int oj); :`
Adds an additional tracer when two tracers are too far away from each other.
input:
  - `**sonda`: coordinates of the tracers that must be covered by the new neighbourhoods, $\vec{t}_{n,i}$.
  - `npu`: number of tracers.
  - `index`: index of the tracer to be added
  - `****pfin`: coefficients of the polynomials obtained in the previous steps of the algorithm.
  - `**info`: information of the centres and radii of the previous neighbourhoods propagated.
  - `inbo`: index of the first neighbourhood propagated in the last integration step.
  - `n`: dimension of the vector field.
  - `oj`: degree of the state variable polynomials.

  output:
  - `**sonda`: coordinates of the tracers for the next step, $\vec{t}_{n+1,i}$.
  - `return`: number of tracers at the end of the adding procedure.

- `int evalpolcon(double *xini, double **info, double ****pfin, int n, int oj); :`
Given the initial condition $x_0+\vec{\xi}$, computes the approximation of $\phi(t;t_0,x_0+\vec{\xi})$ using the polynomials obtained in the splitting neighbourhoods propagation of the flow.
input:

- `*xini`: initial condition that must be propagated.
- `**info`: information table that contains the centres and the radii of the neighbourhoods.
- `****pfin`: coefficients of the polynomials that approximates the flow map through the different steps.
- `n`: number of variables in the system.
- `oj`: degree of the state variable polynomials. $\mathcal{M} = \{x \in \mathbb{R}^n | x]_n = vy0\}$.

output:

- `*xini`: position of the propagation of the initial condition $x_0+\vec{\xi}$, $P_i(P_{i-1}(\dots P_0(\vec{\xi}))) \approx \phi(t; t_0, x_0+\vec{\xi})$.
- `return`: 0.

localinversion.c

In this library there are all the functions related to the inversion of polynomials developed in Section 2.6

- `void locinv_fipome(double ***x, int n, int deg, double ***y);`:
  Given an initial polynomial $P(\vec{\xi})$ which approximates a function f, this function computes the polynomial $Q(\vec{\xi}) \approx f^{-1}$, i.e. such that $P(Q(\vec{\xi})) = \vec{\xi}$ using the fixed point method.
  input:
  - `***x`: coefficients of the polynomial $P(\vec{\xi})$.
  - `n`: number of variables of the function approximated by x.
  - `deg`: degree of the polynomial $P(\vec{\xi})$.

  output:
  - `***y`: coefficients of the polynomial $Q(\vec{\xi})$.
  - `return`: void.

- `void locinv_new(double ***x, int n, int deg, double ***y);`:
  Given an initial polynomial $P(\vec{\xi})$ which approximates a function f, this function computes the polynomial $Q(\vec{\xi}) \approx f^{-1}$, i.e. such that $P(Q(\vec{\xi})) = \vec{\xi}$ using Newton's method.
  input:
  - `***x`: coefficients of the polynomial $P(\vec{\xi})$.
  - `n`: number of variables of the function approximated by x.
  - `deg`: degree of the polynomial $P(\vec{\xi})$.

  output:
  - `***y`: coefficients of the polynomial $Q(\vec{\xi})$.
  - `return`: void.

- `void locinv_rec(double ***x, int n, int deg, double ***y);`:
  Given an initial polynomial $P(\vec{\xi})$ which approximates a function f, this function computes the polynomial $Q(\vec{\xi}) \approx f^{-1}$, i.e. such that $P(Q(\vec{\xi})) = \vec{\xi}$ using recursion method.
  input:
  - `***x`: coefficients of the polynomial $P(\vec{\xi})$.

- `n`: number of variables of the function approximated by x.
- `deg`: degree of the polynomial $P(\vec{\xi})$.

output:

- `***y`: coefficients of the polynomial $Q(\vec{\xi})$.
- `return`: void.

densityprop.c

This library contains the functions necessary to propagate probability density functions through maps.

- `void propadensi(double ***f, int densmod, double ***fdens);`:
  Given a map f, and a density model `densmod`, this function returns the density corresponding to the density model after applying the map f.
  input:
  - `***f`: coefficients of the polynomial approximation of f.
  - `densmod`: density model, by now the only model implemented is a Gaussian distribution, labelled by 2.

  output:
  - `***fdens`: final density function obtained after applying the map f to the PDF given by the density model.
  - `return`: void.

- `void normmusigpoli(double ***x, int n, double *mu, double **sigma, double **f);`:
  Given an initial polynomial $P(\vec{\xi})$ computes the PDF of the normal distribution with mean μ and covariance matrix Σ at $P(\vec{\xi})$.
  input:
  - `***x`: coefficients of the polynomial $P(\vec{\xi})$.
  - `n`: number of variables of the multivariate Gaussian distribution.
  - `*mu`: vector with the mean, μ, of the Gaussian distribution.
  - `**sigma`: covariance matrix, Σ, of the Gaussian distribution.

  output:
  - `**f`: Gaussian distribution with mean μ and covariance matrix Σ at $P(\vec{\xi})$.
  - `return`: void.

A.2 Using openMP to parallelize computations

During the development of this thesis there were some programs that required long computational time. For instance the programs to compute LCS. To compute the LCS several propagations must be done in a grid of points. Each of these computations requires an amount of time. Even if it is not long for one of them, if we must do thousands the CPU time to compute all the integrations may be long.

One possible solution is to use the several processors that computers have nowadays. Then, the CPU work is divided between all the processors and the programs become faster.

The tool: openMP

In order to implement the parallelisation the tool selected is openMP. OpenMP allows to incorporate the parallelisation to our codes. Once the library (openmp.h) is included we need to add before the *for* loop that we want to parallelise the instruction:

```
#pragma omp  parallel for default(none) [other clauses] \
        shared(list1) private(list2)
```

where default(none) indicates that all the variables used inside the loop must be declared as either shared or private. Shared states for those variables (written in list1) which has the same values for all the threads, while private for those variables that are different in each thread. The other clauses states for additional instructions of the openMP that allow to configure other interesting aspects like how the work is distributed through the processors (the so called schedule clause) or how many processors should be used during the parallelisation block.

Observations on the parallelisation

These are some considerations that has been observed while working with the parallelisation.

- Whenever the objective is to parallelise two loops, an inner one and an outer one, for instance if we are interested in parallelise the integration of a grid of points, there are two possibilities. On one hand, it is possible to parallelise each row, or to parallelise the full grid. When implementing the two options, the difference is in which loop the `#pragma` instruction is. The first case corresponds to parallelise the inner loop:

  ```
  //parallelising the inner loop
  for(i=0;i<n;i++){
    #pragma omp  parallel for default(none) [other clauses] \
          shared(i,list1) private(j,list2)
    for(j=0;j<n;j++){
      integration (i,j)
    }
  }//--- end parallelisation ---
  ```

  while the second case corresponds to parallelise the outer loop:

  ```
  //parallelising the outer loop
  #pragma omp  parallel for default(none) [other clauses] \
          shared(list1) private(i,j,list2)
          for(i=0;i<n;i++){
            for(j=0;j<n;j++){
              integration (i,j)
            }
          }//--- end parallelisation ---
  ```

  Figure A.1 shows the difference between both methods. In green and blue for the inner and outer loop respectively. Observe that parallelising the inner loop, the program should divide and rejoin the program for each index of the outer loop. And therefore, it has a higher overhead.

- The second observation is related in how the parallelisation works. If a code is parallelised, in principle, it is desired that all the threads were busy during the parallel block. The problem comes when a thread is much faster than the others because it need less time to do the assigned work. To manage this issue openMP incorporates some instructions in order to control the schedule of the parallel block. Let us assume that we are using d threads in a loop consisting of n iterates. Then we can use the *schedule* clause with one of the following arguments:

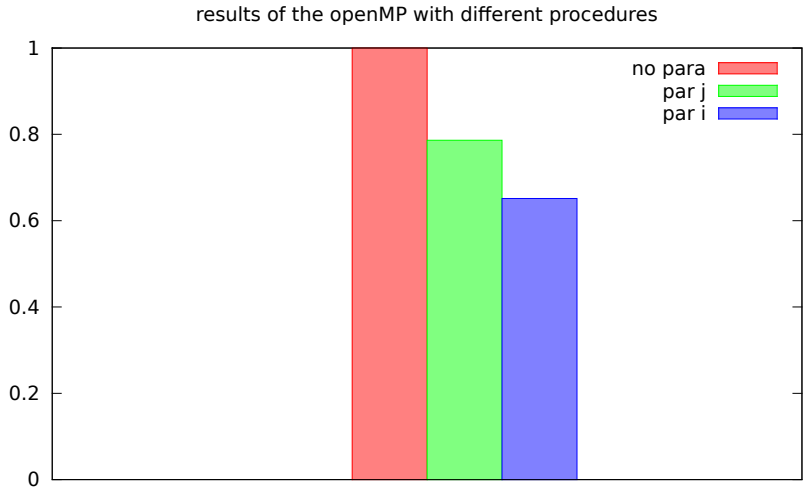


Figure A.1: Normalised Computer time without parallelising (red) parallelising the inner loop (the j loop)(green) or the outer loop (the i loop) (blue). Both parallelisations has been done using two processors.

- static: It is the default schedule. Every thread receives the same number of iterates, in that way the first schedule receives the iterates $1, \ldots, n/d$, the second the iterates $n/d+1, \ldots, 2n/d$, and so on.
- dynamic: Every thread receives the next iterate when the actual iterate is finished. Using that method we ensure that all the threads will be busy almost all the time. But in return we will have some overhead in the assignation of the iterates to the corresponding thread.
- guided: Is the mixture of the previous ones. Every thread receives the next block of iterates when it finishes the actual block. The size of each bloc of iterations is proportional to the number of unassigned iterates divided by the number of threads d.

Figure A.2 (first column) shows how the three methods schedule in a different way the iterates of a procedure.

Which should be the best scheduling? Computing the Clock time we obtain that with static scheduling the computer spend approximately 61 seconds, while using both guided and dynamic scheduling the computer spend 41 seconds. So the dynamic and guided seems to be better than static, but why?
To solve this question we can look at which time is computed every iterate. Figure A.2 (second row) shows which iterates are finished in function of time for the three methods mentioned above. For instance, using the guided scheduling the iterate 700 is approximately done 25 seconds after the beginning of the execution while using the static scheduling this happens approximately 40 seconds later. There we can see that using static scheduling one processor is idle while the other still is computing. So we are loosing a processor because the scheduling is not done correctly.

We are interested into avoid the overhead produced by the dynamic and guided scheduling. A possible way of reduce that fact consist in to evaluate the integrations inside the grid in a random way. To do that we define a permutation vector P (or matrix). Then, we compute the parallelisation using the indices inside the permutation vector instead of the indices of the initial matrix. Figure A.3 shows how the iterates are distributed inside the threads as well as the time needed to reach the ith iterate. There we can see that the iterates are distributed like in the dynamic case but it still need more time (between 1 and 4 per cent more depending in the execution of the program). This situation is due to the fact that the assignation is done randomly, although both processors have more ore less the same amount of work one of them has slightly

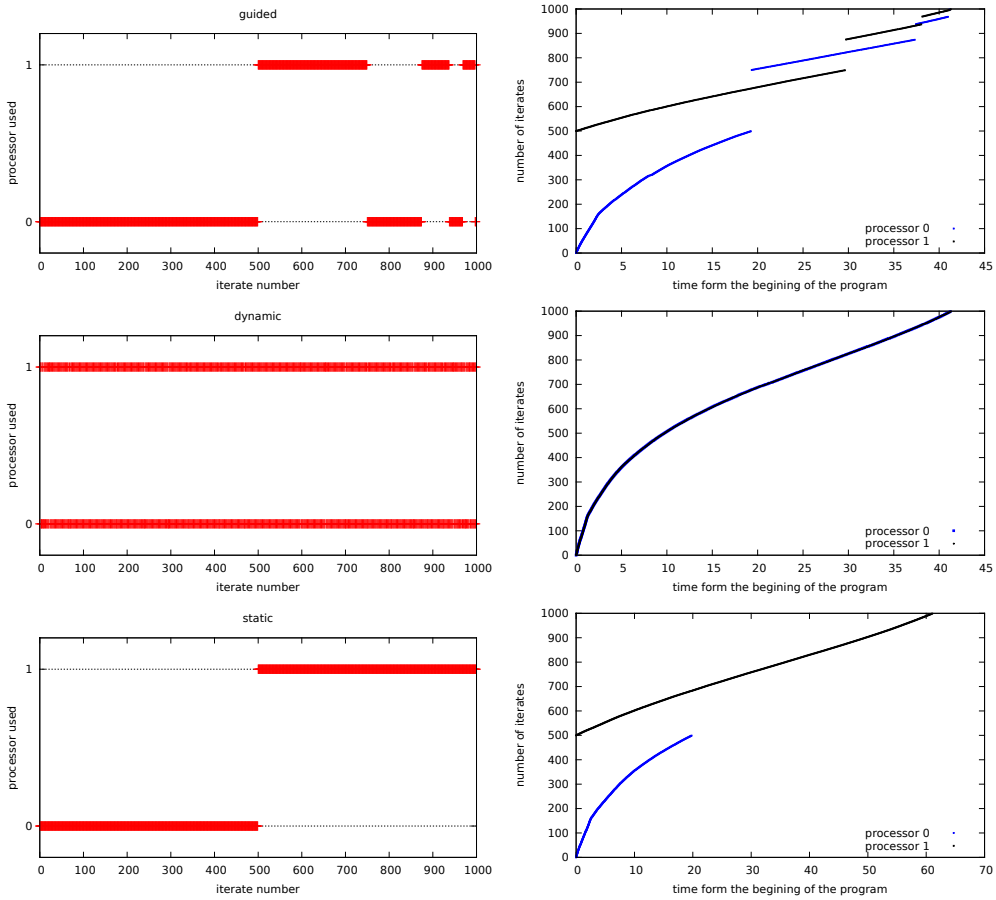


Figure A.2: First column, scheduling of the iterates using (from top to bottom) guided, dynamic and static schedule. Second column: number of iterate in function of the time that need the program to arrive to that iterate for guided, dynamic and static schedule (from left to right).

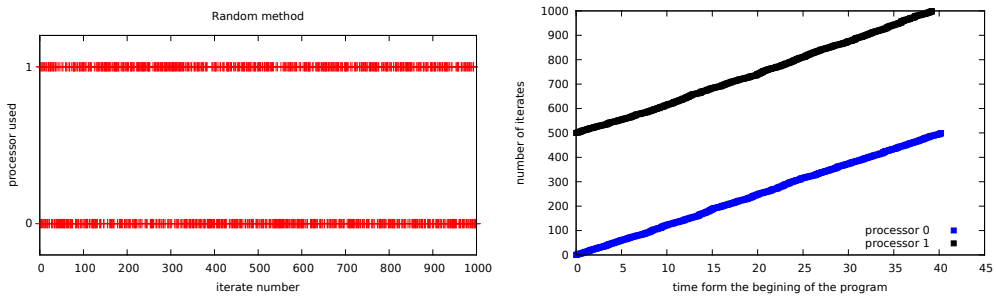


Figure A.3: Left: distribution of the $P(i\text{th})$ iterate inside the processor. Right: ith iterate in function of the time that need the program to arrive to that iterate using the random method.

longer computations. This longer computations do not compensate the overhead produced by the dynamic schedule.

Final remarks: a few things that should be taken into account:

- The declaration of variables in openMP should be done carefully. It is recommended to do not let default variables. All the variables must be declared as shared or private depending on the usage of the variable. Specially one should be careful with:
  - pointer variables,
  - global variables,
  - static variables declared in a function called inside an openMP bloc.

  By default the three types of variables are declared as shared variables. In the case that they take different values on different threads it can ruins our computations. This can be solved by adding special statement: `#pragma omp threadprivate(list)`. This indicates that the variables in *list* are private to each thread.

- A fourth scheduling procedure is available. On it the schedule is assigned during the run-time, allowing the user to change it while executing. To implement that method the function `omp_set_schedule(style,chunk)` should be used. Here, style is a number associated to a certain schedule (1 for static, 2 for dynamic and 3 for guided) and chunk is the minimum length associated to each block. Finally the `schedule(runtime)` instruction should be used at the beginning of the parallel section.

Bibliography

[Be07] Berz, M. and Makino, K., Tech. Rep MSUHEP-060804, Deparment of Physics and Astronomy, Michigan State University, East Lansing, MI 48824. *COSY INFINITY Version 9.0 beam physics manual*, 2007. see also http://cosyinfinity.org.

[AA00] M.R. Akella and K.T. Alfriend. Probability of Collision Between Space Objects. *Journal of Guidance, Control and Dynamics*, 23(5):769–772, 2000.

[ABBR12] A. Abad, R. Barrio, F. Blesa, and M. Rodríguez. Algorithm 924: Tides, a taylor series integrator for differential equations. *ACM Trans. Math. Softw.*, 39(1):5:1–5:28, November 2012.

[ADLBZB10] R. Armellin, P. Di Lizia, F. Bernelli-Zazzera, and M. Berz. Asteroid close encounters characterization using differential algebra: the case of apophis. *Celestial Mechanics and Dynamical Astronomy*, 107(4):451–470, 2010.

[AFJ+08] E.M. Alessi, A. Farrès, À. Jorba, C. Simó, and A. Vieiro. Efficient usage of self validated integrators for space applications. Ariadna final report, contract no: 20783/07/nl/cb, ESTEC (European Space Agency), 2008.

[AFV+09] E.M. Alessi, A. Farrès, A Vieiro, À. Jorba, and C. Simó. Jet transport and applications to neo's. In *Proceedings of 1st IAA Planetary Defense Conference*, ESA Conference Bureau, 2009.

[CGS03] P.M. Cincotta, C.M. Giordano, and C. Simó. Phase space structure of multi-dimensional systems by means of the mean exponential growth factor of nearby orbits. *Physica D: Nonlinear Phenomena*, 182(3–4):151 – 178, 2003.

[CS00] P.M. Cincotta and C. Simó. Simple tools to study global dynamics in non-axisymmetric galactic potentials – i. *Astron. Astrophys. Suppl. Ser.*, 147(2):205–228, 2000.

[DH97] M. Dellnitz and A. Hohmann. A subdivision algorithm for the computation of unstable manifolds and global attractors. *Numerische Mathematik*, 75(3):293–317, 1997.

[dVP10] W.H. de Vries and D.W. Phillion. Monte carlo method for collision probability using 3d satellite models. In *Proceedings of the Adv. Maui Opt. Space Surv. Tech. conference*, 2010.

[FGL97] Cl. Froeschlé, R. Gonczi, and E. Lega. The fast lyapunov indicator: a simple tool to detect weak chaos. application to the structure of the main asteroidal belt. *Planetary and Space Science*, 45(7):881 – 886, 1997.

[FH92] J. L. Foster and S. E. Herbert. A parametric analysis of orbital debris collision probability and maneuver rate for space vehicles. Technical report, NASA Johnson Space Center, August 1992.

[GBM06] J. Grote, M. Berz, and K. Makino. High-order representation of poincarée maps. In M. Bücker, G. Corliss, U. Naumann, P. Hovland, and B. Norris, editors, *Automatic Differentiation: Applications, Theory, and Implementations*, volume 50 of *Lecture Notes in Computational Science and Engineering*, pages 59–66. Springer Berlin Heidelberg, 2006.

[GJMS91] G. Gómez, À. Jorba, J. Masdemont, and C. Simó. Study refinement of semi-analytical Halo orbit theory. ESOC contract 8625/89/D/MD(SC), final report, European Space Agency, 1991. Reprinted as *Dynamics and mission design near libration points. Vol. III, Advanced methods for collinear points*, volume 4 of World Scientific Monograph Series in Mathematics, 2001.

[GJMS93] G. Gómez, À. Jorba, J. Masdemont, and C. Simó. Study of Poincaré maps for orbits near Lagrangian points. ESOC contract 9711/91/D/IM(SC), final report, European Space Agency, 1993. Reprinted as *Dynamics and mission design near libration points. Vol. IV, Advanced methods for triangular points*, volume 5 of World Scientific Monograph Series in Mathematics, 2001.

[GJMS01] G. Gómez, A. Jorba, J.J. Masdemont, and C. Simó. *Dynamics and Mission Design Near Libration Points. Vol. IV Advanced Methods for Triangular Points*. World Scientific, 2001.

[GLMS85] G. Gómez, J. Llibre, R. Martínez, and C. Simó. Station keeping of libration point orbits. ESOC contract 5648/83/D/JS(SC), final report, European Space Agency, 1985. Reprinted as *Dynamics and mission design near libration points. Vol. I, Fundamentals: the case of collinear libration points*, volume 2 of World Scientific Monograph Series in Mathematics, 2001.

[GLMS87] G. Gómez, J. Llibre, R. Martínez, and C. Simó. Study on orbits near the triangular libration points in the perturbed Restricted Three–Body Problem. ESOC contract 6139/84/D/JS(SC), final report, European Space Agency, 1987. Reprinted as *Dynamics and mission design near libration points. Vol. II, Fundamentals: the case of triangular libration points*, volume 3 of World Scientific Monograph Series in Mathematics, 2001.

[GMB05] J. Grote, K. Makino, and M. Berz. High-order validated representation of poincaré maps. In *Proceedings of the 5th WSEAS/IASME Int. Conf on Systems Theory and Scientific Computation*, pages 324–429, September 2005.

[GMDTC09] E. Gawlik, J.E. Marsden, P.C. Du Toit, and S. Campagnola. Lagrangian coherent structures in the planar elliptic restricted three-body problem. *Celestial Mechanics and Dynamical Astronomy*, 103(3):227–249, 2009.

[Hal11] G. Haller. A variational theory of hyperbolic lagrangian coherent structures. *Physica D: Nonlinear Phenomena*, 240(7):574 – 598, 2011.

[Har08] À. Haro. Automatic differentiation tools in computational dynamical systems. Technical report, Universitat de Barcelona, 2008.

[HCF+15] A. Haro, M. Canadell, J.-Ll. Figueras, A. Luque, and J. M. Mondelo. The parameterization method for invariant manifolds: from rigorous results to effective computations. To appear in the Applied Mathematical Sciences Springer book series, 2015.

[HY00] G. Haller and G. Yuan. Lagrangian coherent structures and mixing in two-dimensional turbulence. *Physica D: Nonlinear Phenomena*, 147(3–4):352 – 370, 2000.

[Jor99] À. Jorba. A methodology for the numerical computation of normal forms, centre manifolds and first integrals of hamiltonian systems. *Experimental Mathematics*, 8(2):155–195, 1999.

[JZ05] À. Jorba and M. Zou. A software package for the numerical integration of odes by means of high-order taylor methods. *Experimental Mathematics*, 14(1):99–117, 2005.

[Kau66] W. M. Kaula. *Theory of Satellite Geodesy: Applications of Satellites to Geodesy*. Blaisdell Publishing Company, March 1966.

[Las13] J. Laskar. Is the solar system stable? In Bertrand Duplantier, Stéphane Nonnenmacher, and Vincent Rivasseau, editors, *Chaos*, volume 66 of *Progress in Mathematical Physics*, pages 239–270. Springer Basel, 2013.

[LSM07] F. Lekien, S.C. Shadden, and J.E. Marsden. Lagrangian coherent structures in n-dimensional systems. *Journal of Mathematical Physics*, 48(6):–, 2007.

[MADLBZ15] A. Morselli, R. Armellin, P. Di Lizia, and F. Bernelli-Zazzera. A high order method for orbital conjunctions analysis: Monte carlo collision probability computation. *Advances in Space Research*, 55(1):311 – 333, 2015.

[MG05] O. Montenbruck and E. Gill. *Satellite Orbits: Models, Methods and Applications*. Springer, September 2005.

[MS81] R. McKenzie and V. Szebehely. Non-linear stability around the triangular libration points. *Celestial Mechanics*, 23:223–229, March 1981.

[Oli15] Z.P. Olikara. personal comunication, 2015.

[PTVF07] W.H. Press, S.A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3 edition, 2007.

[SBS+11] C. Sabol, C. Binz, A. Segerman, K. Roe, and P.W. Schumacher Jr. Probability of collision with special perturbation dynamics using the monte carlo method. In *Proceedings of the AAS/AIAA Astrodynamics Specialist Conference*, July-August 2011.

[SH14] C.R. Short and K.C. Howell. Lagrangian coherent structures in various map representations for application to multi-body gravitational regimes. *Acta Astronautica*, 94(2):592 – 607, 2014.

[Sim01] C. Simó. *Global Analysis of Dynamical Systems*, chapter Global dynamics and fast indicators. Taylor & Francis, Bristol, 2001.

[SSST13] C. Simó, P. Sousa-Silva, and M. Terra. Practical stability domains near l4,5 in the restricted three-body problem: Some preliminary facts. In Santiago Ibáñez, Jesús S. Pérez del Río, Antonio Pumariño, and J. Ángel Rodríguez, editors, *Progress and Challenges in Dynamical Systems*, volume 54 of *Springer Proceedings in Mathematics & Statistics*, pages 367–382. Springer Berlin Heidelberg, 2013.

[SV09] C. Simo and A. Vieiro. Resonant zones, inner and outer splittings in generic and low order resonances of area preserving maps. *Nonlinearity*, 22(5):1191–1245, 2009.

[Sze67] V. Szebehely. *Theory of Orbits: The Restricted Problem of Three Bodies*. Academic Press, 1967.

[The14] The PARI Group, Bordeaux. *PARI/GP version* `2.5.3`, 2014. available from `http://pari.math.u-bordeaux.fr/`.

[VAdLL14] M. Valli, R. Armellin, P. di Lizia, and M.R. Lavagna. Nonlinear filtering methods for spacecraft navigation based on differential algebra. *Acta Astronautica*, 94(1):363 – 374, 2014.

[VBS11] R. Vitolo, H. Broer, and C. Simó. Quasi-periodic bifurcations of invariant circles in low-dimensional dissipative dynamical systems. *Regular and Chaotic Dynamics*, 16(1-2):154–184, 2011.

[VS15] J.C. Vallejo and M.A.F. Sanjuán. The forecast of predictability for computed orbits in galactic models. *Monthly Notices of the Royal Astronomical Society*, 447(4):3797–3811, 2015.

[WDLA+15] A. Wittig, P. Di Lizia, R. Armellin, K. Makino, F. Bernelli-Zazzera, and M. Berz. Propagation of large uncertainty sets in orbital dynamics by automatic domain splitting. *Celestial Mechanics and Dynamical Astronomy*, 122(3):239–261, 2015.

[Wil15] D. Wilczak. *CAPD DynSys Library documentation*, 2015.

[YN93] S. Yoden and M. Nomura. Finite-time lyapunov stability analysis and its application to atmospheric predictability. *Journal of the Atmospheric Sciences*, 50(11):1531–1543, 1993.