A Methodology for the Numerical Computation of Normal Forms, Centre Manifolds and First Integrals of Hamiltonian Systems

Àngel Jorba
Departament de Matemàtica Aplicada i Anàlisi
Universitat de Barcelona
Gran Via 585, 08007 Barcelona (Spain)
E-mail: angel@maia.ub.es

December 28th, 1997

Abstract

This paper deals with the effective computation of normal forms, centre manifolds and first integrals in Hamiltonian mechanics. These kind of calculations are very useful since they allow, for instance, to give explicit estimates on the diffusion time or to compute invariant tori. The approach presented here is based on using algebraic manipulation for the formal series but taking numerical coefficients for them. This, jointly with a very efficient implementation of the software, allows big savings in both memory and execution time of the algorithms if we compare with the use of commercial algebraic manipulators. The algorithms are presented jointly with their C/C++ implementations, and they are applied to some concrete examples coming from celestial mechanics.

Keywords: normal forms, centre manifolds, first integrals, algebraic manipulators, invariant tori.

Contents

1 Introduction

The importance of invariant objects in understanding the phase space of a dynamical system is well-known since Poincaré. Invariant objects are not only interesting by themselves but also because they organize the flow nearby. Despite its importance, there are not many numerical methods to compute such objects. The aim of this paper is to explain some techniques that can help to perform some of these computations, in the particular case in which the system is Hamiltonian. As we will see, many topics can be extended to general (analytic) systems and also to discrete dynamical systems. Among the several possible approaches, we have chosen methods based on the computation of (truncated) normal forms and (approximated) first integrals. Truncated normal forms are very useful since they provide, under suitable hypotheses, integrable approximations to the dynamics. The integrable character allows to explicitly give all the invariant objects (like tori) in the phase space. If the normal form approximates the true dynamics, then the invariant objects of the initial system are also approximated accordingly (for examples of this, see [29], [50], [32]). Approximated first integrals are quantities that are almost preserved by the flow of the system. This means that their surface levels are almost invariant by the flow. This property can be used to obtain information about some aspects of the dynamics. For instance, if one is able to estimate the corresponding remainders, then it is not difficult to bound the diffusion velocity around elliptic fixed points (for a numerical example, see [11]).

One of the main problems faced when considering these kind of computations is how "to store" the object in the computer. The easiest case is the computation of a single trajectory, that can be stored as a sequence of points in the phase space. Note that, when the invariant object has bigger dimension, it can be very difficult (usually, it is impossible) to store it by simply storing a net of points. The approach taken here is to use some kind of series expansion (like a power or Fourier expansions, or a combination of both) to represent the object. The advantage is that in many cases only "a few" terms of these series are needed to get a good accuracy and that they can be handled very easily. As disadvantages we note that sometimes they have convergence problems making impossible to represent the object in this way. Due to the particularities of the problems considered here we will only focus on the use of power expansions. You can find examples with trigonometric expansions in [27] and [23]. For a general discussion, see [44] and [49].

Sometimes, when only a qualitative description of the dynamics is needed, it is enough to use a low order computation (this is the typical situation encountered, for instance, in the analysis of a bifurcation). This is not the case considered here. The methodology presented in this paper is directed to produce high order computations, with a high degree of accuracy, ready for use in many practical applications. This necessity usually comes from the applications of the dynamical systems theory to real problems, like the design and analysis of trajectories for some spacecrafts (see [21], [22], [12], [15], [16], [17], [18], [19], [20] and [47]). We also want to mention that, sometimes in academical problems, one needs to perform very accurate computations. In this direction we refer, as examples, to [45] and [48], where the computation (by means of formal expansions) of exponentially small quantities is considered.

Hence, the first point addressed is how to build an efficient algebraic manipulator (in a quite efficient language like C or C++) in order to handle these expansions in a very fast way, and using as little memory as possible. Then, as an application, we will use these routines to study some aspects of the Restricted Three Body Problem (RTBP). More concretely, we will show how to use these techniques to describe the dynamics near the five equilibrium points of the RTBP. The paper also discusses related topics like error analysis (including the use of interval arithmetic), efficiency (both from the memory and speed points of view) and some possible extensions (more variables, time dependence, etc.) to these routines. The source code for (almost) all the algorithms explained here can be retrieved from the web server `http://www.maia.ub.es/dsg`, in the "preprints and publications" section.

In this work we have made extensive use of the particularities of Hamiltonian systems, so many of the algorithms explained here can not be used outside of this environment. However, the methodology we use to build algebraic manipulators is very general and can be applied in a lot of different contexts. To facilitate the reading, in Appendix A we have included a summary of the main concepts and properties of Hamiltonian systems.

In the next sections we summarize a few problems to justify the necessity of this kind of computations. Of course, there are many other applications of these tools (both practical and theoretical) beyond the ones presented here. We have selected a few simple ones to have concrete problems to work with, and to be able to give concrete results. We hope that the interested reader will not have problems in applying these ideas to similar problems in other fields.

In order to simplify the exposition, we restrict ourselves to analytic and autonomous Hamiltonian systems with three degrees of freedom (3DOF), having a fixed point at the origin. In Section 7 we will discuss possible extensions to more general contexts.

1.1 Examples

Now let us give a few problems where classical numerical methods (i.e., numerical integrations of single trajectories) are not enough to give a good answer. They will be used as examples along this paper. As it has been mentioned before, we assume some knowledge of Hamiltonian mechanics.

1.1.1 Dynamics near an elliptic equilibrium point

Let us assume that we are interested in the dynamics near an elliptic equilibrium point (that, for simplicity, we will locate at the origin) of a three degrees of freedom Hamiltonian system. Note that, as the phase space is of dimension six, it is very difficult to get a "picture" of the dynamics using numerical integration of single trajectories.

Assume we are able to rewrite the initial Hamiltonian H as

$$H = H_0 + H_1, \tag{1}$$

where H_0 is an integrable Hamiltonian (so, in this case, the phase space is completely foliated by invariant tori) and H_1 is a non integrable one. Then, if H_1 is small enough near the point, the trajectories corresponding to H_0 are close to the trajectories of H (at least for moderate time spans). Hence, from the integrable character of H_0 it is not difficult to obtain approximations for the invariant tori of H. The effect that H_1 has on the solutions of H_0 is discussed in Appendix A. Essentially one has that, near the origin, most of the tori of H_0 are not destroyed by H_1 but only slightly deformed. However, it is generally accepted that H_1 can create some trajectories that escape from any small vicinity of the origin, making this point unstable. This phenomenon is usually called diffusion, and it was first noticed by V.I. Arnol'd in [3].

Let us assume that we are also interested in estimates of the diffusion time near the origin. Note that computational effort needed to do this by single numerical integration is too big that it can not be considered a feasible option: the big number of trajectories one has to consider plus the huge time interval of integration (this also introduces the problem of accumulation of rounding errors) for each one makes this calculation impossible for present computers. An alternative procedure can be the following: let us assume that we are able to rewrite the initial Hamiltonian H as in (1). As H_0 is integrable, the diffusion present in H must come from H_1. Hence, one can easily derive bounds for the diffusion time in terms of the size of H_1. Of course, in order to produce realistic diffusion times one needs to have H_1 as small as it can be. A standard way of producing the splitting (1) is by means of a normal form calculation: H_0 is the normal form and H_1 the corresponding remainder (see [14] and also [43] and [28]).

There are alternative ways of estimating the diffusion time near elliptic equilibrium points. For instance, one can construct approximate first integrals near the point and estimate the "drift" of these integrals. Of course, although one can use as many first integrals as degrees of freedom, it is enough to use a single positive-definite integral (near the point, its level surfaces split the phase space in two connected components so they act as a barrier to the diffusion).

We want to note that although from the theoretical point of view both approaches are equivalent (the first integrals we compute are in fact the action variables of the normal form), from the computational point of view they behave differently. We will see this in detail later on.

1.1.2 Dynamics in a centre manifold

Let us consider a 3DOF Hamiltonian system with an equilibrium point at the origin. Assume that the linear flow around this point is of the type centre×centre×saddle.

We are interested in finding a description of the dynamics in a neighbourhood (as big as possible) of the origin. One possibility is to perform the so-called reduction to the centre manifold. That is, to perform changes of variables in order to uncouple (up to some finite order) the hyperbolic behaviour from the centre one (one can look at this as a partial normal form). Hence, the restriction of the Hamiltonian to this (approximate) centre manifold will be a 2DOF Hamiltonian system. So, selecting an energy level $H = h$

and doing a suitable Poincaré section we can produce a collection of 2-D plots that can give a good description of the dynamics. This was first used in [15] (see also [26]).

1.2 Methodology

Here we will present the methodology we use to deal with those computations, based on the use of algebraic manipulators. There are several possible schemes, depending on the kind of calculation we are interested in. For instance, if the procedure only needs to substitute trigonometric series in the nonlinear terms of the equations (like in the Lindstedt-Poincaré method, see [15], [23] and [26]), one of the best choices is to look for a recurrent expression of those nonlinear terms (the substitution is simply done by inserting the series into the recurrence). In this paper, we will apply schemes that work with the power expansion of the Hamiltonian (when the system is not Hamiltonian, one must work with the differential equations –or with the equations of the map if the system is discrete– but, of course, this increases the computational effort). So a general scheme for the problems considered here is the following:

1. Power expansion of the Hamiltonian around the origin.
2. Complexification of the Hamiltonian. This is not a necessary step but, as we will see, it allows to simplify further computations.
3. Changes of variables (usually by means of Poisson brackets), up to some finite order.
4. Realification of the final Hamiltonian. Again, this is not a necessary step. It is done only to reduce the size of the resulting series.
5. Computation of the change of variables that goes from the initial Hamiltonian to the final one.

So, one needs computer routines for all these steps. A natural way of handling the power expansions is as a sequence of homogeneous polynomials:

$$H = \sum_{k\geq 2} H_k,$$

where H_k is an homogeneous polynomial of degree k. So one of the most important problems will be to deal with homogeneous polynomials of several variables. As we will see, the bottleneck (according to speed) of the methods exposed here is given by the speed we can manage homogeneous polynomials.

1.3 Previous computer packages

There are several computer packages that, in principle, are able to deal with the computations mentioned here. Among the commercial software may be the most well-known packages are Maple and Mathematica. They have the advantage of being very general packages (they can deal with much more problems than the ones exposed here) but, on the other hand, they are not very efficient (both in time and memory) in each concrete case. So if one is interested in low order computations (sometimes this is enough in academic problems) they can be considered as a valid option. But if one wants to reach high orders, commercial packages are not competitive at all. In this case one has to go to software very adapted to the particularities of the problem to take advantage of them. This is the line we have followed in this work. In fact, it is possible to write even faster routines (see Section 5) but then the code is, in our opinion, more obscure. In some cases (especially when you take into account the developping –and debugging– time), the gain in speed is not big enough to justify the loss of clarity. Anyway, we hope the reader interested in this point will not have problems in modifying the software.

There are some other packages similar to this one in the literature. In fact, there is a long tradition in the Celestial Mechanics field about building algebraic manipulators. We cite, among others, [13], [24], [7], [6], [40], [8] and [34] (see also references therein). These works are directed to concrete problems of Celestial Mechanics and are very efficient when dealing with them.

1.4 Programming considerations

The programming language used here is ANSI C, except when we have had to use complex numbers. In this case we have used the capability of C++ to overload the arithmetic operators with the complex operations. It is not strictly necessary to know C or C++ to read this paper but, to see the details of the implementation of the algorithms, it is necessary to look at the source code. Details about C and C++ programming can be found in the standard references [33] and [51].

If the reader does not have (and does not want to have) a C++ compiler, it is not difficult (but tedious) to rewrite these operations in order to have an ANSI C source code. Another interesting possiblity is to use the SCC package (see [41]). This is, essentially, a preprocessor that allows to define new operations and overload with them the standard arithmetic operators (in a similar way that C++ does). SCC translates this code into C, to be processed by a standard C compiler.

It is not difficult to use these algorithms in other languages. In fact, the first version of these routines was written in Fortran 77. The main advantage (in our opinion) that C gives in this problems is the dynamic allocation of memory and the use of structures. In Fortran 77 one has to set some parameters before compiling in order to declare big enough arrays, to have room for the expansions as well as working space.

The paper is structured as follows: Sections 2 and 3 give the details on the algebraic manipulators, Section 4 is devoted to some applications of this software to concrete problems, Section 5 discuss the efficiency of these methods as well as some improvements to them. Section 6 contains some remarks about the propagation of the rounding errors. Finally, Section 7 points out some extensions to these methods to be used in more complex problems (for instance, when the Hamiltonian depends on time in a periodic or quasiperiodic way). We have added a couple of Appendices in order to make the paper self contained: Appendix A contains a short description of the properties of Hamiltonian systems used here and Appendix B contains a basic description of the Restricted Three Body Problem, as well as some properties that are used in the applications. We have included these appendices because we are not aware of similar summaries in the literature.

2 Basic Tools

In this section we will describe the basic algorithms and routines used to handle homogeneous polynomials. This is the most important part of the package. In what follows, to simplify the notation, we will assume that the set of the natural numbers, $\mathbb{N}$, contains 0.

2.1 Storing and retrieving monomials

Let us assume that we want to store an homogeneous polynomial P_n of degree n, with 6 variables $(x_0, \ldots, x_5)$,

$$P_n = \sum_{\substack{k \in \mathbb{N}^6 \\ |k|=n}} p_k x^k,$$

where we use the notation $x^k \equiv x_0^{k_0} \ldots x_5^{k_5}$ and $|k| = k_0 + \cdots + k_5$. For the moment we assume that all the coefficients p_k are different from zero. Let us define $\psi_6(n) = \#\{k \in \mathbb{N}^6$ such that $|k| = n\}$ (that is, $\psi_6(n)$ denotes the number of monomials of P_n).

To store the polynomial we use an array of $\psi_6(n)$ components (the kind of array depends on the kind of coefficients of the polynomial), and we use the position (index) of a coefficient inside the vector to know the monomial it corresponds to. To this end we will construct a function (let us call it `llex6`) that, given a place inside the array (that is, an integer between 0 and $\psi_6(n) - 1$) it returns the multiindex that corresponds to this coefficient. Of course we need the inverse function (we call it `exll6`) to know where to store a given monomial.

Before going into the details of these functions, we want to stress that they are the most important ones: if they are efficient, the package will be efficient. This will be discussed in Section 5.2.

Let us go back to the software. In order to have a fast implementation, we use an integer array (we assume here that every integer is four bytes long) to store some information to be used by function `llex6`. This array has $\psi_6(n)$ components and each one contains (encoded) the multiindex of the corresponding coefficient. We use this array in the obvious way: each time we need to know the exponent of the monomial whose coefficient is stored in the place j of the homogeneous polynomial, we get it from the component j of this array.

The way of encoding the multiindex k is the following: as we know the degree we are working with, one of the exponents (say k_0) is redundant, so we only need to store $k_1, \ldots, k_5$. This has to be stored inside a 32 bits number, so we can use 6 bits for each index, leaving 2 unused. This introduces the restriction $k_j < 64$. As we want to handle homogeneous polynomials the maximum degree allowed is 63, more than enough for the applications done here.

2.2 The routines

Here we go into the details of the basic routines of the manipulator. Their source code is stored in the file `mp6.c`. As these routines are the most important ones, we will discuss them more carefully. For portability reasons, in the heading of several files we redefine the standard type `int` as `integer`. This is because the first version of these routines was developed in an old 286 machine (where ints were 2 bytes long) and it was run on a HP workstation (ints were 4 bytes long). So, if we work with the type `integer` we can control the kind of integers used (simply by redefinig this type). Of course, this is not relevant for present (1997) computers.

Headings of the file `mp6.c`

Here we have placed the declarations of three variables that must be accessible by all the routines in this file. They are named `nor`, `clmo` and `psi`, and are initialized by routine `imp6`. Their meaning is explained in the next sections.

Routine `imp6`

This routine has to be called before using any other routine in the package, because it allocates and initializes some internal arrays to store the encoded multiindices. The only parameter of this routine is an integer (`nr`) that contains the maximum degree we want to use. This value is stored in the variable `nor`.

Before continuing, let us define the function $\psi_i(n)$ as

$$\psi_i(n) = \#\{k \in \mathbb{N}^i \text{ such that } |k| = n\},$$

that can be easily evaluated by means of the recurrence

$$\psi_i(n) = \sum_{j=0}^{n} \psi_{i-1}(j) = \binom{n+i-1}{i-1}. \tag{2}$$

The routine starts doing a couple of checks to verify that we are calling it with a suitable degree and that the `integer` type of the machine (or compiler) is long enough.

The first step is to allocate space to store the values of the function $\psi_i(j)$. At this moment we only need to know ψ_6 but we will also compute $\psi_2, \dots, \psi_5$ (they will be needed later on). To this end we allocate a rectangular matrix `psi` with the first index ranging from 2 to 6 and the second one from 0 to `nor`. Then, the values $\psi_i(j)$ are computed (using the recurrence given in (2)) and stored in the position (i, j) of the matrix `psi`.

Next step is to allocate space for the table `clmo`. The first dimension of this table ranges from 0 to `nor`, and it refers to the degree of the homogeneous polynomials. If the first index is i, the second index ranges from 0 to $\psi_6(i) - 1 \equiv$ `psi[6][i]` -1. The position

(i, j) of this array is the encoded version of the multiindex of the monomial number j of a polynomial of degree i. Once this table has been allocated, we have to fill it with the information about the multiindices. First of all we define an order inside the set of multiindices of a given degree: Let k be a multiindex of degree n and let us define $\overline{k}$ as the integer number (in base $n+1$) $k_5k_4k_3k_2k_1k_0$ (for instance, if $k = (1,2,3,4,5,6)$ then $\overline{k} = 654321$). Then, the order is given by

$$k^{(1)} < k^{(2)} \Leftrightarrow \overline{k^{(1)}} < \overline{k^{(2)}}.$$

This is usually called reverse lexicographic order. Now, for a given degree `i`, we compute all the multiindices according to this order and we store them in the table `clmo`: the first one for degree `i` is `(i,0,0,0,0,0)`, and all the others are generated by routine `prxk6` (see below). We store the components of each multiindex in the corresponding place of `clmo`, using 6 bits for each component: this means that the coded version of the multiindex is (note that we do not code k_0 because, as we know the degree, it is redundant)

$$k_1 + k_2 \times 2^6 + k_3 \times 2^{12} + k_4 \times 2^{18} + k_5 \times 2^{24}. \tag{3}$$

This is the value we will store in `clmo[i][j]`, where we have assumed that `j` stands for the place of the multiindex (and the monomial) inside this order.

Finally, the routine returns the amount of memory (in Kbytes) used by these tables. It is up to the calling routine to print this value.

Routine `amp6`

It frees the memory allocated by `imp6`. Of course, once it has been called the manipulator can not be used until a new call to `imp6` has been done.

Routine `llex6`

Given a place `lloc` and a degree `no`, it computes the multiindex corresponding to them. The way it works is very straightforward because the multiindex is contained (encoded) in `clmo[no][lloc]`, and to decode it we only need to invert (3) using the modulus function. An improvement for this routine consists in directly extracting the corresponding bits from `clmo[no][lloc]`.

Routine `exll6`

Given a multiindex `k` of degree `no` (this is redundant information but it is very useful to avoid calling these routines in a wrong way), it returns the corresponding place. So, this is the inverse of `llex6`. The implementation of this routine can be done in many ways. Let us see the one we have used here. Let us denote by $k = (k_0, \ldots, k_5)$ the multiindex and let n be $k_0 + \cdots + k_5$. Define $k^{(5)}$ as $(k_0, \ldots, k_4)$ and let $n_5 = n - k_5$ be the degree of $k^{(5)}$. Then, if we are able to compute

1. the number of multiindices $(\ell_0, \dots, \ell_5)$ of 6 variables with degree n such that $0 \le \ell_5 < k_5$,
2. the place it corresponds to $k^{(5)}$ among the multiindices of 5 variables of degree n_5,

then, the sum of these two quantitites is the place we are looking for. The first of these numbers is $\psi_5(n_5+1) + \cdots + \psi_5(n)$, and can be easily obtained from the table `psi`. The second one is the same problem we want to solve, but with one dimension less, so we can apply again the same procedure until we reach dimension 2 (this corresponds to polynomials of two variables), where the solution of the problem becomes obvious. An improvement for this routine is to use auxiliar tables to reduce these integer computations.

Routine `ntph6`

This routine returns the number of monomials of a given degree (this information is contained in the array `psi`).

Routine `prxk6`

It is used to produce all the multiindices of a given order, according to the order we are using. For more details, we refer to the source code.

2.3 Taking advantage of symmetries

It is quite common in physical examples to have some kind of symmetry in the Hamiltonian. For instance, in the examples used in this paper we have a symmetry with respect to the variable z (see (4) and Appendix B). This implies that not all the possible monomials of the power expansion of the Hamiltonian are really present. In the examples used here we have that, if i is the exponent of z and j the exponent of p_z, the only monomials that appear in the expansion are the ones in which $i+j$ is even. Hence, taking this into account it is possible to reduce the amount of memory used and the computing time by an approximate factor of two.

In order to exploit the symmetry we have developped special versions of the routines of Section 2.2. The source code is stored in the files `mp6s.c` and `mp6p.c`.

File `mp6s.c` contains the same routines as `mp6.c` (but with an "`s`" at the end of the name, to be able to use them in the same program if necessary), but assuming that the only monomials present are the ones that satisfy that $k_4 + k_5$ is even. As they work in a very similar way, we only mention the main differences:

`imp6s` Function $\psi_6(n)$ is not longer valid to compute the number of monomials, because of the symmetry. The number of monomials for a given degree n is now given by

$$\sum_{j=0}^{[\frac{n}{2}]} (2j+1)\psi_4(n-2j),$$

where $[\frac{n}{2}]$ denotes the integer part of $n/2$.

`exll6s` To have a simple formula for the position corresponding to a given index, we have changed the order used for the monomials: we first use the reverse lexicografic order for the exponents (k_4, k_5) and, in second place, the reverse lexicographic order for the exponents (k_0, k_1, k_2, k_3). This is usally called product reverse lexicographic order. It allows to easily derive a closed formula for the position (see the source code).

`prxk6s` It is changed in order to produce the exponents in the product reverse lexicographic order defined above.

File `mp6p.c` contains the same routines as `mp6s.c`, but with a different symmetry: here it is assumed that all the monomials that are present satisfy that $k_4 + k_5$ is odd (this kind of symmetry will appear in some computations, see Section 4.4). The implementation is almost identical to the one of `mp6s.c`, so we do not add further remarks.

In fact, as the examples considered in this paper have the above mentioned simmetry, we do not make use of the routines in `mp6.c`. We have included them for the sake of completeness, and because they are the most natural ones to start describing how these kind of routines work.

Finally, let us note that if the symmetries are "too complex" to derive closed formulas for the routines `exll`, one can always perform a binary search on the array `clmo`. In this case, it is very convenient to use an order such that the integer values stored in `clmo` are sorted as integer numbers. Although this is not as efficient as a closed formula, it can be easily applied in all the cases.

2.4 Different number of variables

As the examples in this paper are three degrees of freedom Hamiltonian systems, the basic routines explained here handle polynomials with six variables. If one is interested in a different number of variables, it is not difficult to build the corresponding basic routines. For instance, in Section 4.1.3 we need to handle the normal form of a 3DOF Hamiltonian system, that depends on 3 variables. To this end it is very easy to write the corresponding routines, using the same algorithms as for six variables. We have put those routines in file `mp3.c`, Note that this file is, essentially, a minor modification of file `mp6.c`. In a similar way we have derived the routines of `mp4s.c` and `mp4p.c`, that are needed during the reduction to the centre manifold (see Section 4.3).

3 Handling Homogeneous Polynomials

The routines of this section are contained in the files `basop6s.cc` and `basop6sp.cc`. Note that we have several versions of some of them, in order to deal with polynomials with different symmetries. As before, we recommend to give a look at the source code, since it will clarify (we hope!) our explanations.

3.1 Sums and products

Let `p1` and `p2` be two arrays containing (the coefficients of) homogeneous polynomials of degrees `g1` and `g2`.

Let us assume that both polynomials are of the same degree and that we want to add them, storing the result in an array called `p3`. If we call `nm` the number of monomials of one of these polynomials (this is the value returned by a routine like `ntph6`), then the sum is easily computed:

```
for (i=0; i<nm; i++) p3[i]=p1[i]+p2[i];
```

Here we have assumed that we have defined the operation `+` for the type of the coefficients of the polynomial: if they are `double` variables one does not need to do anything special since they are already defined in any C compiler. If they are of `complex`[1] type, we assume that we are working in C++ or that we are using a C extension able to overload the arithmetic operators (like [41]) with the `complex` operations. If the coefficients are more sophisticated types, we assume that we have the corresponding arithmetic, as well as a way of overload the arithmetic operators.

Note that, in a similar way, it is very easy to implement the product of a `complex` number by a polynomial, so we avoid any comment on that.

Let us see the product of homogeneous polynomials (now we are not assuming that `p1` and `p2` have the same degree). The algorithm is very straightforward and uses the rutines explained in Section 2: Let us call n_1 and n_2 the number of monomials of each polynomial `p1` and `p2`. Then, to multiply the monomial number i of `p1` with the monomial number j of `p2` we only have to compute the corresponding multiindices $k^{(i)}$ and $k^{(j)}$, to ask for the position where the coefficient of the monomial $k^{(i)} + k^{(j)}$ must be stored, and to add there the product of the coefficients. Doing this for all the possible values of i and j we obtain the desired product. You can give a look at the source code for more details.

3.2 Poisson bracket

The Poisson bracket of two homogeneous polynomials can be implemented using the same ideas as the product. The algorithm we have used is based on the following identity:

$$\left\{ \sum_{k,\ell} p_{k,\ell} x^k y^\ell, \sum_{k',\ell'} q_{k',\ell'} x^{k'} y^{\ell'} \right\} = \sum_{k,\ell,k',\ell'} p_{k,\ell} q_{k',\ell'} \left(\sum_{j=1}^{3} (k_j \ell'_j - k'_j \ell_j) \frac{x^{k+k'} y^{\ell+\ell'}}{x_j y_j} \right),$$

where, of course, k, ℓ, k' and ℓ' belong to $\mathbb{N}^3$. Thus, for any term of this sum, we proceed as in the product of homogeneous polynomials: we look first for the exponents of the monomials involved, then we compute the exponents of the resulting monomials and, finally, in the position corresponding to those monomials, we add the resulting coefficients. For more details, look at the source code.

[1] Unless otherwise specified, `complex` means a structure with two members of type `double`: the real and the imaginary part

3.3 Input and output

We have coded several routines in order to read and write power expansions and homogeneous polynomials (both in ASCII and binary format). We are not providing a complete set of routines to handle all the possible situations, but we simply give the ones needed in the examples. As we have mentioned before, our intention is to show that they can be written very easily and we hope that the interested reader will not have any problem in coding any similar routine.

You will see that there are a lot of different routines, each one for a different purpose. Although it is not difficult to write a common front-end for all of them we have not done so. The main reason is that the aim of this paper is not to give an easy-to-use library of functions but to show how to build such a library. Hence, we have avoided any construction that hides the inner working of the routines.

3.3.1 ASCII files

There are several routines to read and write homogeneous polynomials and series. The format is very easy: for each coefficient, we compute the corresponding exponents and we write the exponents followed by the value of the coefficient. We use a single line for each coefficient.

There are several sets of routines for the different kind of series (`mp6s`, `mp6p`, real or complex coefficients, etc.). Some of the routines use a threshold to decide if a monomial has to be written or not (if the absolute value of the coefficient is smaller than the threshold, the monomial is not written).

The advantage of ASCII files is that they can be printed and read by an ordinary text editor. The main disadvantage is that they are very big and that they are written and read very slowly. Hence, they are only used to write the final results and to store intermediate values during the developping/debugging stages.

3.3.2 Binary files

This format is used to store intermediate calculations or series that are only used as an input for other programs (like the changes of variables).

The routines that write homogeneous polynomials simply write (sequentially) all the coefficients in the file, without storing the exponents of the corresponding monomials. The reading routine will read all the coefficients in a row, without any checking (except, of course, the end of file), and they will be stored sequentially in the corresponding array. Each coefficient is then identified by its position inside the file. This is to minimize the size of the file and to maximize the speed at which the file is handled.

The routines that write series simply write sequentially the homogeneous polynomials, adding a little bit of information to the file according to the kind of series stored. This extra information is put at the beginning and consist of four integer values, with the following meaning:

1. The first integer contains the number of variables of the expansion.

2. The second integer contains the kind of simmetry of the expansion. This value can be:

   `0`: No symmetry. All the monomials are present in the file.

   `1`: Symmetry of '`s`' kind: all the monomials such that the sum of the exponents of the last two variables is odd are missing.

   `2`: Symmetry of '`p`' kind: all the monomials such that the sum of the exponents of the last two variables is even are missing.

3. The third integer is the initial degree of the expansion (usually, it is 1 or 2).

4. The fourth integer is the final degree of the expansion.

The reading routine checks this information and gives the corresponding error messages when necessary. Note that there is nothing indicating the kind of coefficients of the stored series. It is up to the user to take this into account.

Of course, writting in this way assumes that the reading routine will use the same algebraic manipulator as the writting routine, since the exponent of a monomial is known from the position of the monomial inside the series. You have to take this into account if you modify these routines.

4 Examples

In this section we are going to apply these routines to perform some practical computations on a concrete model. For this purpose we have selected the well-known Restricted Three Body Problem (RTBP), near one of the five equilibrium points $L_{1,\ldots,5}$ of the system. For a basic description of this problem, see Appendix B.

4.1 Example I: Normal form

The Hamiltonian H of the RTBP, in suitable adimensional units and with the origin at L_5, takes the form

$$H = \frac{1}{2}(p_x^2 + p_y^2 + p_z^2) + yp_x - xp_y + \left(\frac{1}{2} - \mu\right) x \mp \frac{\sqrt{3}}{2} y - \frac{1-\mu}{r_{PS}} - \frac{\mu}{r_{PJ}}, \tag{4}$$

where $r_{PS}^2 = (x - x_S)^2 + (y - y_S)^2 + z^2$, $r_{PJ}^2 = (x - x_J)^2 + (y - y_J)^2 + z^2$, $x_S = 1/2$, $y_S = \mp\sqrt{3}/2$, $x_J = -1/2$ and $y_J = \mp\sqrt{3}/2$. The "$-$" sign is for L_4 while "$+$" is for L_5. The mass ratio is taken below the Routh critical value, so the origin is linearly stable.

4.1.1 Complexification and power expansion

The first step is to produce a power expansion of (4) up to a finite order N,

$$H = \sum_{n=2}^{N} H_n,$$

where H_n denotes a homogeneous polynomial (in six variables) of degree n. To describe how to produce such expansion, let us focus first on the term $1/r_{PS}$ of (4). Naming ψ the angle between $(x_S, y_S, 0)$ and (x, y, z), and being $\rho^2 = x^2 + y^2 + z^2$ one has

$$\frac{1}{r_{PS}} = \frac{1}{\sqrt{1 - 2\rho\cos\psi + \rho^2}} = \sum_{n=0}^{\infty} \rho^n P_n(\cos\psi),$$

where P_n is the Legendre polynomial of degree n. Let us define A_n as $\rho^n P_n(\cos\psi)$ (note that A_n is an homogeneous polynomial of degree n). Then, from the well-known recurrence of the Legendre polynomials one obtains

$$A_{n+1} = \frac{2n+1}{n+1}(xx_S + yy_S)A_n - \frac{n}{n+1}(x^2 + y^2 + z^2)A_{n-1}, \tag{5}$$

starting with $A_0 = 1$ and $A_1 = xx_S + yy_S$. Note that this recurrence can be easily implemented using a routine that multiplies homogeneous polynomials. Moreover, as the computational effort is not very high and it is numerically stable, this recurrence is very suitable for a practical computation. Of course, the expansion of $1/r_{PJ}$ can be done in the same way, and the remaining terms of (4) can be added directly to the sum of these two expansions.

Before continuing, let us make a very important remark. As the first step is to put H_2 in normal form (see Section B.1), and this is done by a linear change of variables, we can insert this change of variables directly into the recurrence (5), in order to produce the expansion with this first change already done. This is much better than to compose the change with the final expansion. Note that the real normal form of H_2 is

$$H_2 = \frac{\omega_1}{2}(x^2 + p_x^2) + \frac{\omega_2}{2}(y^2 + p_y^2) + \frac{1}{2}(z^2 + p_z^2),$$

where we have kept the same notation for the variables and we have used that the frequency in the vertical direction is always 1 (for all μ). In order to facilitate the computation of the generating function, it is very convenient to "diagonalize" H_2 (see Section A.4 for more details). This can be done by a (complexifying) change of variables:

$$x = \frac{q_1 + \sqrt{-1}p_1}{\sqrt{2}}, \qquad p_x = \frac{\sqrt{-1}q_1 + p_1}{\sqrt{2}}, \tag{6}$$

and similar expressions for the other variables. So, we compose this change with the first one to obtain a (complex and symplectic) linear change of variables that brings the initial H_2 into the normal form

$$H_2 = \sqrt{-1}\omega_1 q_1 p_1 + \sqrt{-1}\omega_2 q_2 p_2 + \sqrt{-1} q_3 p_3. \tag{7}$$

This is, in fact, the change inserted into the recurrence (5) to produce the expansion in these variables.

The routines that perform this expansion are contained in the file `exp-l5.cc`. Let us give a short description of them.

`ccvl5` It computes the change of variables that put the initial H_2 into the final normal form (7). This change is derived in Section B.1.

`exp_l5` This is the main routine for the expansion of the Hamiltonian. It calls `exrec` and `reste`.

`exrec` This routine performs one of the recurrences (5). It is called twice by `exp_l5` (first to expand $1/r_{PS}$ and then $1/r_{PJ}$).

`reste` This routine computes the terms in (4) that are neither $1/r_{PS}$ nor $1/r_{PJ}$.

4.1.2 The normal form

The next step is the computation of the normal form. We use Lie series, since they are very suitable to perform explicit computations. More details on this method are contained in Sections A.3 and A.4, and here we will only focus on the implementation. The main properties of the Poisson bracket used here are that it is bilinear and that, if P_r and Q_s are homogeneous polynomials of degrees r and s respectively, then $\{P_r, Q_s\}$ is an homogeneous polynomial of degree $r+s-2$.

The computation is done in several steps, one for each degree. Let us explain the first of these steps. We want to compute a generating function G_3 (an homogeneous polynomial of degree 3) such that the transformed Hamiltonian

$$H' = H + \{H, G_3\} + \frac{1}{2!}\left\{\{H, G_3\}, G_3\right\} + \frac{1}{3!}\left\{\left\{\{H, G_3\}, G_3\right\}, G_3\right\} + \cdots, \tag{8}$$

has no terms of degree 3. Using that $H = H_2 + H_3 + H_4 + \cdots$ one obtains that the terms of degree 3 of the transformed Hamiltonian H' are

$$H'_3 = H_3 + \{H_2, G_3\}.$$

Hence, we ask $H'_3 = 0$. This equation is easily solved, because H_2 is of the form (7): let us denote by k^q the three indices of k that correspond to the variable q and by k^p the ones of p. The expressions of H_3 and G_3 can be written as

$$H_3 = \sum_{|k|=3} h_3^k q^{k^q} p^{k^p}, \qquad G_3 = \sum_{|k|=3} g_3^k q^{k^q} p^{k^p}.$$

Hence, assuming that the frequencies $\omega = (\omega_1, \omega_2, 1)$ of H_2 are rationally independent, it is not difficult to obtain the coefficients g_3^k of G_3:

$$g_3^k = \frac{-h_3^k}{\sqrt{-1}\,\langle k^p - k^q, \omega\rangle}.$$

As in this case $|k|$ is odd, the denominator $\langle k^p - k^q, \omega\rangle$ is never zero. When $|k|$ is even one must consider the case $k^p = k^q$ (note that, as the components of ω are rationally independent, this is the only possiblity to produce a zero divisor). This implies that this monomial can not be elliminated and then we select the corresponding g_3^k equal to zero. Of course, if one wants to perform the normal form up to degree N, it is enough to ask $\langle k, \omega\rangle \neq 0$ when $0 < |k| < N$. If this condition is not satisfied we can still perform a resonant normal form, that is, we can elliminate all the monomials except the ones for which $\langle k, \omega\rangle = 0$ (usually called resonant monomials). Even when the frequencies are rationally independent, some of the denominators $\langle k, \omega\rangle$ can be very small, reducing drastically the domain where these transformations are valid. In this case is also possible to leave those monomials in the normal form, in order to keep a reasonable size for the domain of convergence (note that then the normal form will not be integrable, see [43] for a discussion of this technique).

Once the generating function has been computed, we can use (8) to compute the transformed Hamiltonian. Let us see the implementation we have used for this formula. Assume we are working with an expansion of H up to degree N:

$$H = H_2 + H_3 + \cdots + H_{N-1} + H_N,$$

and, for instance, we want to transform it using as a generating function an homogeneous polynomial G_3 of degree 3. To save memory, the result will be stored in the same space used for H. To give the idea, let us write explicitly the firsts steps of the method:

step 1.1 $H_N \leftarrow H_N + \{H_{N-1}, G_3\}$

step 2.1 $H_{N-1} \leftarrow H_{N-1} + \{H_{N-2}, G_3\}$

step 2.2 $H_N \leftarrow H_N + \frac{1}{2!}\left\{\{H_{N-2}, G_3\}, G_3\right\}$

step 3.1 $H_{N-2} \leftarrow H_{N-2} + \{H_{N-3}, G_3\}$

step 3.2 $H_{N-1} \leftarrow H_{N-1} + \frac{1}{2!}\left\{\{H_{N-3}, G_3\}, G_3\right\}$

step 3.3 $H_N \leftarrow H_N + \frac{1}{3!}\left\{\left\{\{H_{N-3}, G_3\}, G_3\right\}, G_3\right\}$

$\vdots$

Note that the Poisson bracket done in step **2.1** can be re-used to compute step **2.2**, the one in **3.1** can be used in **3.2** and this last one in **3.3**, and so on. In this way, we are minimizing the number of arithmetic operations (each Poisson bracket is done only once), we can work on the initial Hamiltonian (the parts of it that are overwritten are not needed in further steps) and the need of working space is not very big: we need working space for two homogeneous polynomial of degree N in the worst case (one is used to store the Poisson bracket done in **i.j-1** to be used in **i.j**, the other one is to compute the next Poisson bracket). This has been implemented in routine `traham` (see below).

The routines for these algorithms are contained in file `nf6s.cc`. Let us give a short description of them:

nf6s The main routine for the computation of the normal form. It assumes that the initial Hamiltonian H_2 is in diagonal form. The routine gets the frequencies ω from the corresponding places of H_2 and, for each degree, it computes the generating function of the change of variables (see `cage`) and transforms the Hamiltonian (see `traham`). The generating function is written in a binary file, degree by degree. As this is not considered a series but a sequence of different generating functions, no heading is added to the file (this heading was explained in Section 3.3.2).

cage Computes the generating function corresponding to a given degree. One of the parameters is a pointer to a function that, given the exponents of the monomial, returns 1 if the monomial has to be removed from the normal form, and 0 otherwise. This is done in this way in order to facilitate to change the "killing criterion".

traham This transforms the Hamiltonian according to the algorithm mentioned above, using the generating function computed in `cage`. After the transformation, the routine puts zero in the places that corresponds to killed monomials. This lines can be commented if the user does not want to do that. In this case, those values will not be exactly zero because of the rounding errors (see Section 6 for a more detailed discussion).

Moreover, in the file `kill-nf.c` there is the function that decides if a given monomial has to be killed or not (see remarks in routine `cage` above).

4.1.3 Back to real coordinates

The final step is to realify the transformed Hamiltonian. The case of seminormal forms can be done using the considerations in Section 4.4 (see also Section 4.3).

Let us start by using the inverse of the complexifying change (6),

$$q_j = \frac{x_j - \sqrt{-1}y_j}{\sqrt{2}}, \quad p_j = \frac{-\sqrt{-1}x_j + y_j}{\sqrt{2}}, \quad j = 1,2,3, \tag{9}$$

where we use q_1, q_2, q_3, p_1, p_2 and p_3 for x, y, z, p_x, p_y and p_z respectively. In order to put the Hamiltonian in the easiest possible form, we compose this change with

$$x_j = \sqrt{2I_j}\cos\phi_j, \quad y_j = -\sqrt{2I_j}\sin\phi_j, \quad j = 1,2,3.$$

This is equivalent to

$$q_j = I^{1/2}\exp(\sqrt{-1}\phi_j), \quad p_j = -\sqrt{-1}I^{1/2}\exp(-\sqrt{-1}\phi_j), \tag{10}$$

Hence, as the monomials that appear in the normal form have the same exponent both for positions and momenta ($k^q = k^p$ in the notation above), the change (10) makes them to depend only on the actions I_j:

$$h_k q^{k^q} p^{k^p} = h_k(\sqrt{-1})^{|k^q|} I^{k^q}.$$

Routines in file `rnf6s.cc` apply the change (10) to the normal form. As we have to deal with polynomials of three variables, we need the routines of `mp3.c`. Now, let us give a brief description of `rnf6s.cc`.

`rnf6s` Applies the change (10) to the normal form. It assumes that the manipulator contained in `mp3.c` has been initialized by the calling routine.

`check_rlf` This is to check if a given multiindex corresponds to the normal form. It is used by `rnf6s` to know which ones are the terms to realify (all the others are assumed to be zero).

4.1.4 Main program and results

A main program that uses these routines is contained in the file `main_nf.cc`. It is a very short and easy to read program that computes the normal form, up to a given order, around the equilibrium point L_5 of the RTBP. The output of the program is contained in several files: the normal form is stored in the ASCII file `nf.res`, the generating function is stored in the binary file `nf.gen` and the linear change of variables used to diagonalize the linearized vectorfield around L_5 is put in the ASCII file `nf.cvl`. The parameters used in the actual run (the degree and the mass parameter) are stored in the ASCII file `nf.ctl`.

In Table 1 we include the first terms of the normal form for the Earth-Moon case. The last column in that table corresponds to the imaginary part of the coefficients and it should be zero. It is not zero due to the rounding errors in this process. This column is not taken into account for subsequent computations with the normal form, but we have included it to give an heuristic estimate about how roundoff errors behave in this case. See also remarks in Section 6.2.

4.2 Example II: First integrals

Let us assume that we are interested in computing (approximate) first integrals of a given Hamiltonian system H, in a neigbourhood of an equilibrium point. Of course, if H is not integrable, the first integrals will not be convergent but, close enough to the equilibrium point, they will be quantities that are almost preserved by the flow. This can be used for different purposes, for instance to bound the diffusion time around an elliptic equilibrium point. We refer to Section A.7 for more details.

Let us summarize the procedure to compute those integrals. Let $H = \sum_{j\geq 2} H_j$ the power expansion of H around the equilibrium point (that for simplicity we assume is the origin), where each H_j is an homogeneous polynomial of degree j. Let us denote by $F = \sum_{j\geq 2} F_j$ the expansion for the first integral we are looking for. Then, as F must satisfy $\{H, F\} = 0$, one has the following recursive equation

$$\{H_2, F_n\} = -\sum_{j=3}^{n} \{H_j, F_{n-j+2}\}, \tag{11}$$

1	0	0	9.5450087346985146e-01	0.0000000000000000e+00
0	1	0	-2.9820811951603865e-01	0.0000000000000000e+00
0	0	1	1.0000000000000000e+00	0.0000000000000000e+00
2	0	0	1.1568661352624510e-01	1.9950987004677088e-15
1	1	0	-1.7127952377596927e+00	1.6464553654140052e-14
0	2	0	3.3855424993051031e-01	-1.6132819906367057e-14
1	0	1	8.9130919974620498e-02	7.6519569570206439e-16
0	1	1	2.2531870698905809e-01	-1.8153505446248392e-15
0	0	2	-2.2354591332438556e-03	-9.4980345474466460e-17
3	0	0	-2.9478784724938123e-01	-8.8195876408494016e-14
2	1	0	8.1656946590496773e+00	-2.4411186045905709e-11
1	2	0	-5.4586887250177915e+02	-1.5117692624804247e-10
0	3	0	-5.1021278394561250e+01	-4.4683867166008548e-11
2	0	1	-4.3799694571855952e-01	1.4016167918231831e-13
1	1	1	1.4116984215354037e+01	-9.8915697135260128e-12
0	2	1	2.0187058976961225e+00	-1.7373839789087187e-12
1	0	2	-5.5905039470536266e-02	-1.9157633989231475e-14
0	1	2	-1.7898209821803412e-01	1.0442695912981926e-14
0	0	3	-5.1325740689130392e-05	-1.8243944685427148e-15
4	0	0	1.2775512804655591e+00	-6.7431830234291555e-10
3	1	0	-3.5068853734061122e+01	-5.4267568786972957e-10
2	2	0	-5.4875008796056733e+04	4.6093383107028067e-08
1	3	0	3.2223469268329442e+04	1.5779814576740623e-07
0	4	0	3.5185007412806153e+03	-7.6035412461354353e-09
3	0	1	2.1759346547114546e+00	-4.0412062928307053e-10
2	1	1	2.0101335538551211e+01	-1.8846523533034277e-09
1	2	1	1.3647631576893851e+04	1.2205347940204729e-08
0	3	1	1.4507386615262367e+03	-1.4038343557712580e-09
2	0	2	2.1938211094638973e+00	-6.7723532456943524e-11
1	1	2	-4.9540209943972513e+01	6.1719014476874278e-10
0	2	2	-1.0178742459873320e+01	3.9061093991945888e-11
1	0	3	3.5475354854384022e-02	8.1934049924464103e-13
0	1	3	7.1211245121958200e-02	1.7453335694916767e-11
0	0	4	5.2188851777046352e-04	3.2941657400195349e-13

Table 1: This table contains the coefficients of the normal form for the Earth-Moon case ($\mu = 1.2150581623433623 \times 10^{-2}$). The first three columns contain the exponents of the actions, and the fourth and fifth columns are the real and imaginary part of the coefficients. Imaginary parts must be zero, but they are not due to the rounding errors (see more comments in the text).

that puts F_n in terms of F_{n-1}, ..., F_2 and H. To simplify the discussion, let us assume that H_2 is in complex diagonal form, that is, $H_2 = \sum_j \sqrt{-1}\omega_j q_j p_j$. As $\left\{q_j p_j, q^\ell p^\ell\right\} = 0$, we have that

1. the coefficients of the monomials $q^\ell p^\ell$ of F_n can not be determined,
2. if the coefficient of the monomial $q^\ell p^\ell$ in the right-hand side of (11) is not zero, this equation can not be solved.

There are conditions under which the right-hand side of (11) does not contain monomials of the form $q^\ell p^\ell$. For instance, when the frequencies are nonresonant ($\langle k, \omega\rangle = 0 \iff k = 0$) and the initial Hamiltonian is reversible (i.e., an even function of the momenta).

The example we are going to use is again the RTBP near $L_{4,5}$ for the Sun–Jupiter case, for which the frequencies are nonresonant.[2] As in this case the Hamiltonian is not reversible, we need another kind of argument to justify the solvability of equation (11). Here we will use (without proof) that this equation can be solved for the RTBP case, and that it is enough to take zero the terms of F_n that we can not determine ($q^\ell p^\ell$). We refer to [11] for a discussion of these properties.

Another point worth to mention is that F_2 is not determined by the method, but it should be selected by the user. In [11], as they want to have 3 first integrals $F^{(j)}$, $j = 1, 2, 3$, they use $F_2^{(j)} = \sqrt{-1} q_j p_j$, $j = 1, 2, 3$. We note that, if one only wants to bound the diffusion around the point, it is enough to compute a single definite-positive first integral. This can be achieved using, for instance, $F_2 = \sum_j \sqrt{-1} q_j p_j$. Of course, one can put different "weights" in front of each $q_j p_j$ to try to optimize the size of the region of effective stability (we recall that this region is, in general, not spherical).

4.2.1 Implementation

Note that most of the routines needed for this case have already been developed for the normal form computation. In fact, we only need to implement the recursion (11) and the realification of the (approximate) first integral.

An overall of the program is the following. First we expand the Hamiltonian around the equilibrium point using the same rutines as in the normal form case (the ones of the file `exp-l5.cc`). In this way we obtain a complexified expansion such that the second degree terms are in diagonal form. Then, we solve recurrently equation (11), where the initial value F_2 is provided by the user (this is done by the routines of the file `fi.cc`). Once the first integral has been computed up to the desired order, it is realified (the routines for this are in the files `irex.cc` and `re6s.cc`, and the realification process will be explained in Section 4.5) and written to the ASCII file `fi.res`. This is the only file produced by this program. The main program that controls this process is in `main-fi.cc`.

[2] As in the normal form case, we only need the nonresonance condition up to a finite order. Hence, this is a condition that can be checked in practical examples.

4.3 Example III: Centre manifolds

Let us consider the dynamics near one of the collinear points $L_{1,2,3}$ of the RTBP. We recall that the linearization of the vectorfield at these points is of the type centre×centre×saddle. In order to give an accurate description of the dynamics in a neighbourhood of $L_{1,2,3}$ one can perform the so-called reduction to the centre manifold. This process is explained with more detail in Section A.6 and the idea is the following: let us assume that the diagonal form of H_2 is

$$H_2 = \lambda q_1 p_1 + \sqrt{-1}\omega_2 q_2 p_2 + \sqrt{-1}\omega_3 q_3 p_3, \qquad \lambda, \omega_2, \omega_3 \in \mathbb{R}.$$

Hence, the hyperbolic direction is given (at first order) by the variables (q_1, p_1). Let us perform canonical transformations on the Hamiltonian (in the same way it has been done in Section 4.1.2) but now, instead of cancelling all the nonresonant monomials, we only cancel monomials such that the exponent of q_1 is different from the exponent of p_1 (for a different scheme that cancels less monomials, see [46]). Then, after a finite number of transformations, the Hamiltonian takes the form

$$H = H^{(0)}(q_1 p_1, q_2, p_2, q_3, p_3) + R(q_1, p_1, q_2, p_2, q_3, p_3),$$

where $H^{(0)}$ is the part of the Hamiltonian that we have arranged and R denotes the remainder. As $H^{(0)}$ depends on the product $q_1 p_1$ we can perform the change $I_1 = q_1 p_1$ to produce

$$H = H^{(0)}(I_1, q_2, p_2, q_3, p_3) + R(I_1, \varphi_1, q_2, p_2, q_3, p_3),$$

where φ is the conjugate variable of I_1. If we drop the remainder R (it is very small near the origin) then I_1 is a first integral of the system and putting $I_1 = 0$ we are skipping the hyperbolic part of the Hamiltonian $H^{(0)}$. The resulting two degrees of freedom Hamiltonian represents the flow inside the (approximation to the) centre manifold. So, near the origin, the phase space of the original Hamiltonian must be the phase space of $H^{(0)}(0, q_2, p_2, q_3, p_3)$ times an hyperbolic direction. To visualize the phase space of $H^{(0)}$ one can fix the value of the Hamiltonian and then use a Poincaré section. Varying the value of the Hamiltonian we will obtain a collection of 2-D plots representing the dynamics in the phase space. This has already been done in [15], [26] and [27].

4.3.1 Implementation

The implementation is very similar to the one of the normal form, with the only difference that now we want to kill less monomials. Hence, for the computation of the complex normal form we have used exactly the same routines as before (the ones contained in the file `nf6s.cc`), only changing the function used to decide which monomials are killed (this function is stored in the file `kill-nf.c` for the normal form case and now is the one in the file `kill-cm.c`).

The main difference appears when we need to realify the transformed Hamiltonian. In the normal form case, realification is done by taking advantage ot the particularities of a complete normal form. Here it is a little bit more difficult. Let us summarize the process. First, to save memory, the (still complex) partial normal form is written in a binary file and then it is read monomial by monomial. For each monomial corresponding to the centre manifold[3] (otherwise the monomial is discarded) we compute the result of applying the realifying change (9) to this monomial. The process is the same one used in Section 4.5 (see there for more details), but for four variables monomials. The realified monomials are added to the realified series (different complex monomials can contribute to the same realified monomial) until all the complex monomials are transformed. The routines that perform the realifying process are stored in the files `irex.cc` and `rcm6s.cc`. Finally, the centre manifold is written to an ASCII file. The main program for this computation is stored in the file `main-cm.cc`.

The output files are: `cm.res` contains (in ASCII format) the Hamiltonian reduced to its centre manifold, `cm.gen` is a binary file with the generating function used, `cm.cvl` is an ASCII file with the linear change used to put H_2 in diagonal form and `cm.ctl` contains the parameters used in the actual run.

4.4 Changes of variables

An important part of the computations is to produce the changes of variables going from the final coordinates (normal form or centre manifold) to the initial ones. This can be used for several purposes, ranging from estimates on the diffusion time to the practical computation of invariant tori (of any dimension). We refer to [32] for examples of this.

The global change is split in two different sub-changes. The first one is the linear change that puts H_2 in diagonal form (we will refer to these coordinates as "diagonal" coordinates) plus the translation of the origin from the libration point to the centre of masses of the RTBP. The second sub-change consists of the nonlinear change that goes from the normal form (or centre manifold) coordinates to the diagonal ones. Here we will focus on this last change since the first one is explicitly given in Appendix B.

The process to obtain the nonlinear change is the following. Let us start by considering the first change of variables done on the Hamiltonian by means of a generating function G_3. The corresponding change for this transformation can be obtained by applying the transformation (8) to a single coordinate q_i or p_i $(1 \leq i \leq 3)$,

$$q_i^{(3)} = q_i + \{q_i, G_3\} + \frac{1}{2!}\{\{q_i, G_3\}, G_3\} + \cdots, \tag{12}$$

$$p_i^{(3)} = p_i + \{p_i, G_3\} + \frac{1}{2!}\{\{p_i, G_3\}, G_3\} + \cdots, \tag{13}$$

where $q_i^{(3)}$, $p_i^{(3)}$ denotes the series obtained in this transformation. This is done using the algorithm explained in Section 4.1.2. Note that expressions (12) and (13) are changes of coordinates: they relate the coordinates of the transformed Hamiltonian under G_3 (they are q_i, p_i) with the initial (diagonal) coordinates $q_i^{(3)}$, $p_i^{(3)}$. This idea can be used to

[3]Those are the monomials such that the exponent of q_1 is equal to the exponent of p_1.

produce the changes to higher orders. For instance,

$$\begin{aligned} q_i^{(4)} &= q_i^{(3)} + \left\{q_i^{(3)}, G_3\right\} + \frac{1}{2!}\left\{\left\{q_i^{(3)}, G_3\right\}, G_3\right\} + \cdots, \\ p_i^{(4)} &= p_i^{(3)} + \left\{p_i^{(3)}, G_3\right\} + \frac{1}{2!}\left\{\left\{p_i^{(3)}, G_3\right\}, G_3\right\} + \cdots, \end{aligned}$$

is the transformation that goes from the normal form coordinates of degree 4 to the initial diagonal coordinates. Of course, this transformation is done on the expressions (12) and (13) as if they were Hamiltonians, by means of the algorithm explained in Section 4.1.2. In this way, we obtain the explicit transformation that puts the Hamiltonian in normal form up to the desired order. Note that, when doing these transformations, it is only necessary to transform up to the same degree as in the normal form.

Let us note that the obtained series are still in complex coordinates. They are realified using the methods that will be explained in Section 4.5.

The change corresponding to the centre manifold has some differences with the change for the normal form case. As the centre manifold is of dimension four (the first two variables have been set to zero), the final change is given by six real expansions, each one depending on four variables (the first four expansions are of the type `mp4s` and the last two are of the type `mp4p`).

4.4.1 The inverse change

As before, we are going to focus on the nonlinear part of the change, since the linear part is easily inverted. We only provide routines for the normal form case (the inverse change for the centre manifold can be produced similarly).

This computation is based on the following fact: the change induced by the generating function G is the inverse of the change induced by the generating function $-G$. This is because the change is the time one flow of the Hamiltonian G, and to reverse the time in this flow one has to change the sign of the vectorfield, i.e., of the Hamiltonian G. Hence, one can use the same scheme as before but using as generating functions $-G_n$, $-G_{n-1}$, ..., $-G_4$, $-G_3$, in this order. We refer to the previous section for more comments.

As before, the obtained series are still in complex coordinates. Section 4.5 deals with the algorithms used to realify them.

4.5 Realification of power expansions

A common operation at the end of these computations is the realification of the complex power expansions obtained, because we are usually interested in the dynamics corresponding to real coordinates. Hence, realified expansions are much smaller (the memory needed to store them is halved) and this implies that all the computations involving them are also faster. We want to stress that it is not compulsory to perform such realification, because all the computations with these expansions can be done with the complexified version. The realification is only used for efficiency reasons.

Now let us explain the algorithm used. To simplify the discussion, let us assume we have to realify a 6 variables expansion, in which all the variables have been previously complexified. Note that it is possible to have a complex expression in which not all the variables have been complexified (see, for instance, the expansion of the Hamiltonian in Section 4.3). To start, let us focus on the realification of a single monomial,

$$c_k q_1^{k_1} p_1^{k_2} q_2^{k_3} p_2^{k_4} q_3^{k_5} p_3^{k_6}. \tag{14}$$

Then, in order to apply the realifying change (9), let us make the following remarks:

1. If we know the realification of the product $q_1^{k_1} p_1^{k_2}$, for any k_1 and k_2, we know the realification of all the products $q_2^{k_3} p_2^{k_4}$, $q_3^{k_5} p_3^{k_6}$ (the only difference is in the subindices of the variables).

2. If we know the realifications of the three couples $q_j^{k_{2j-1}} p_j^{k_{2j}}$ $(j = 1, 2, 3)$, the product of these realified expansions (note that each one of them is an homogeneous polynomial with two variables) is not difficult to compute, since we are multiplying polynomials that depend on different variables.

Hence, we will apply the following scheme: first we will compute the realifications of all the powers $q_1^{k_1} p_1^{k_2}$, where the exponent (k_1, k_2) is such that $0 < k_1 + k_2 \leq n$, and n denotes the degree up to which we plan to realify. The result of each realification will be stored in a table (see below). Then, for each monomial like (14), we will obtain from the table the realifications of the three couples $q_1^{k_1} p_1^{k_2}$, $q_2^{k_3} p_2^{k_4}$ and $q_3^{k_5} p_3^{k_6}$ (they will be three homogeneous polynomials of degrees $k_1 + k_2$, $k_3 + k_4$ and $k_5 + k_6$, respectively). Finally, we will form the product (14), taking advantage of the fact that the three homogeneous polynomials depend only on two variables, and that these variables are different. Let us explain this with more detail.

4.5.1 The realifying table

Now we consider the problem of computing and storing expressions like $q^i p^j$, $i \in \mathbb{N}$, $j \in \mathbb{N}$, where

$$q = \frac{x - \sqrt{-1}y}{\sqrt{2}}, \quad p = \frac{-\sqrt{-1}x + y}{\sqrt{2}}. \tag{15}$$

Let us start by the storing procedure. Let us fix i and j, and let us define $m = i + j$. Then, the substitution of (15) into $q^i p^j$ produces an homogeneous polynomial of degree m, in the variables x and y. A natural way of naming the different coefficients of this polynomial is to use a single integer to denote the monomial we refer to: monomial number 0 will be $x^m y^0$, monomial number 1 will be $x^{m-1} y^1$, and so on. Generically, the monomial number k will be $x^{m-k} y^k$, $0 \leq k \leq m$. Note that we need three indices (i, j, k) to identify one of these coefficients (i, j refer to the monomial $q^i p^j$, and k refers to the position of the coefficient inside the realification of $q^i p^j$). Hence, we can look at all these realifications as polynomials with three variables: the coefficient number k of the realification of $q^i p^j$ is the coefficient of the monomial (i, j, k) of a (real but not homogeneous) polynomial of degree

$2m$. This implies that, to store all these realifications, it is enough to allocate space for a three variables power expansion up to degree $2n$, where n denotes the maximum degree we plan to realify. Note that not all the monomials of this expansion are going to be used, but

1. the amount of memory used by the whole table is not very big (see examples below),
2. in this way the access to the elements of the table is very easy (we can use the manipulator `mp3` explained before) and very fast.

It would be possible to only allocate the elements we really need, but this would decrease the speed of the program and, as it has been said before, the amount of memory saved is not enough (in our opinion) to justify the increase in complexity of the program.

To simplify (and to speed up) the computation of the realifying table we also initialize a couple of auxiliar tables, one with the negative powers of $\sqrt{2}$ and another one with the binomial coefficients. With these auxiliar tables, it is not difficult to compute the different powers $q^i p^j$ and to store them in the corresponding place of the table.

The routines that initialize the realifying process have been stored inside the file `irex.cc`. They are the following:

`ini_real` This routine allocates space for the table that will contain the realifications of the different monomials $q^i p^j$. It also computes and stores that table. This rutine calls routine `imp3` (file `mp3.c`) to initialize the tables needed to handle power expansions with three variables.

`end_real` This routine frees the space allocated by `ini_real`, including a call to `amp3` to free the space allocated by `imp3`.

`coef` This routine computes the coefficient of the monomial $x^{k-j}y^j$ in q^k or p^k.

4.5.2 The main algorithm

Now it is not very difficult to realify a power series. In order to minimize the amount of RAM[4] used, the series to be realified is first written in a (binary) file. Then, this file is read sequentially and each monomial is realified and added to the (proper place of the) resulting series.

So, the only point that needs to be discussed is the realification of a single monomial. The process is as follows. Let us use the same notation as in (14). Note that each couple $q_i^{k_j} p_i^{k_{j+1}}$ becomes, once realified, an homogeneous polynomial of degree $k_j + k_{j+1}$ in two variables, x_i and y_i. The coefficients of this polynomial are stored in the suitable places of the realifying table (see Section 4.5.1). Therefore, in order to multiply these three realified polynomials, we will use three (nested) loops to "run" over the coefficients of them (these coefficients are directly obtained from the realifying table). In this way we will obtain the

[4] If this word means nothing to you, ask to your system manager. You should know about the maximum amount of RAM you can use without collapsing your computer.

coefficients of the realification of (14) as the product of these three coefficients with the coefficent c_k. The exponent that corresponds to this final product is easily obtained and this allows to add the coefficient to the suitable place of the resulting series.

4.5.3 The final output

Before continuing with the description of the algorithm let us explain, up to now, what we have obtained. As before, to simplify the discussion we will focus on a couple position-momentum that we will denote as q_1, p_1. Let us denote the initial change of variables that we want to realify as

$$\begin{aligned} q_1' &= q_1 + O_2(q,p), \\ p_1' &= p_1 + O_2(q,p), \end{aligned}$$

where the "primed" variables are the initial ones and the "unprimed" variables the final ones. Of course, $O_2(q,p)$ denotes the higher order terms of the change that we do not write explicitly. After the realification process we have just described, we obtain something like

$$\begin{aligned} \frac{x_1' - \sqrt{-1}y_1'}{\sqrt{2}} &= \frac{x_1 - \sqrt{-1}y_1}{\sqrt{2}} + O_2(x,y), \\ \frac{-\sqrt{-1}x_1' + y_1'}{\sqrt{2}} &= \frac{-\sqrt{-1}x_1 + y_1}{\sqrt{2}} + O_2(x,y). \end{aligned}$$

The next (and final) step is to isolate $x_1' = x_1'(x,y)$ and $y_1' = y_1'(x,y)$. For instance, x_1' can be isolated from the first equation by taking real parts and multiplying by $\sqrt{2}$, and y_1' can be obtained by the first equation by taking the imaginary parts times $-\sqrt{2}$. A similar process can be applied to the second equation to obtain the same expressions. May be the most important conclusion we can get from this fact is that it is enough to compute only one of the expressions for the change of variables: for instance, to obtain the changes of variables for the normal form of Section 4.1 (a 3DOF Hamiltonian) we only need to compute the changes for the three positions. The changes for the three corresponding momenta are obtained from them when realifying (note that we are using that we have complexified with respecto all the variables). Of course, we have taken advantage of this property in the software.

4.5.4 A few remarks

In some cases, it is necessary to realify not all the variables, but only some of them. A typical example appears when we have been dealing with an expansion of the kind centre$\times$saddle. The saddle variables does not need to be complexified, since they already appear in "diagonal form" (see Section 4.1.1). Hence, once the computation is finished, they are still in real form. Of course, the realifying change have to be only applied to the couples q_i, p_i that have been complexified. The main difference appears in the change that corresponds to variables that have not been complexified. Let us denote by q_1, p_1

one of these couples. After the realification (of the complexified variables), the change for q_1, p_1 looks like

$$\begin{aligned} x_1' &= x_1 + O_2(x, y), \\ y_1' &= y_1 + O_2(x, y). \end{aligned}$$

We have changed q_1, p_1 by x_1, y_1 to denote that the realification has been done. Note that the realifying changes have been applied to variables q_j, p_j, $j \neq 1$ (they only affect to $O_2(x, y)$). Hence, we have directly the change of variables (in particular, all the imaginary parts of the coefficients of this change must vanish), without need of taking real or imaginary parts. The bad news are that now we need to compute both changes (for x_1' and y_1'), since we can not derive easily one from another.

4.6 The linear part of the change

We have seen how to produce the nonlinear change for variables used to achieve the normal form but, to reach the initial coordinates we still need to apply the linear change used at the beginning to put H_2 in normal form. This change has been computed in order to diagonalize the second degree terms of the Hamiltonian, and it has been stored in a file. In principle, this transformation goes from the "diagonal" coordinates of H_2 to the usual coordiantes of the RTBP centreed at the equilibrium point. If one is interested in the inverse change, it is not difficult to see that the inverse of any symplectic matrix M can be obtained as $M^{-1} = -JM^\top J$, that is very suitable for numerical purposes.

4.7 Tests of the software

We have done some checks on the software, to be sure that there are no bugs present. The tests we have done are very similar for the three examples so we will mainly focus on the tests for the normal form computation.

To this end, we have written the program `ninf`, that produces a numerical integration of the normal form obtained. In fact, as the normal form is integrable, this program computes the gradient of the normal form for the given actions to obtain the frequencies and then it simply tabulates the solution. Then, this table is sent through the changes of variables into the synodical coordinates of the RTBP. Finally, program `rtbp` tests this table in the following way: for each point of the table, it integrates (numerically) the point to obtain a prediction for the following point of the table. Then, the program writes the differences between the two points (the one obtained from the changes of variables and the one obtained using numerical integration). Ideally, if the normal form, the changes of variables and the numerical integration were all exact (zero error), these differences must be zero. Of course, they are not zero due to the several sources of error.

Let us illustrate this. We have taken the initial conditions $I_1 = I_2 = I_3 = \lambda_0$, with initial phases $\phi_1 = \phi_2 = \phi_3 = 0$, for $t = 0$ (let us call u_0 to this initial condition). We have tabulated the corresponding solution at $t = 0.1$ (let us call u_1 to this vallue), and we have sent both points to synodical coordinates, to obtain two points v_0 and v_1. Then,

λ_0	$\|v_1 - v_0^1\|_2$	λ_0	$\|v_1 - v_0^1\|_2$
0.00001	2.4828078245222093e-16	0.00008	3.4023375555581652e-10
0.00002	5.1198523403369423e-15	0.00016	8.8211434435268124e-08
0.00004	1.3192410121093586e-12	0.00032	2.3101212284736493e-05

Table 2: Differences between a normal form prediction and a numerical integration. The local error of the numerical integration is of the order of 10^{-16} and the normal form (and the corresponding changes of variables) have been computed up to degree 16.

we have computed (numerically) the trajectory of the RTBP that starts at v_0, till $t = 0.1$ (let us call this point v_0^1), with a local error of the order of the roundoff of the arithmetic. The difference $\|v_1 - v_0^1\|_2$ is given in Table 2.

Note that the parameter λ_0 is, essentially, the distance from the initial condition to the origin. If the software is working properly, the error $\|v_1 - v_0^1\|_2$ is due to the truncation of the power series (to degree 16, in the case corresponding to Table 2). Hence, the error should behave like $c\lambda_0^n$, being n the last order in the normal form that we have taken into account (see below). Then, one has that the order of the error can be approximated by

$$n \approx \frac{\ln\left(\frac{e_1}{e_2}\right)}{\ln\left(\frac{\lambda_0^{(1)}}{\lambda_0^{(2)}}\right)}.$$

Applying this to the results in Table 2 we obtain Table 3. The first value in this table is not very accurate because the estimation of the error is not realistic for $\lambda_0^{(1)} = 0.00001$ (it is smaller than 10^{-16} and this is not detected since we are working with double precision arithmetic). The other values are more accurate and produce an exponent for λ_0 that is very close to 8. Note that if the order of the normal form in the (q, p) variables is 16, in the Poincaré variables (see (10)) is 8. Moreover, note that the numerical integrations are done on the differential equations (that involve the derivatives of the Hamiltonian). This means that the error for this case is not of the order of the neglected terms of the Hamiltonian but of the neglected terms of the corresponding differential equations. Hence, as λ_0 "moves" in the space of the Poincaré coordinates, we expect and estimated exponent of the same order as the biggest degree present in the normal form expressed in Poincaré variables.

The same procedure can be applied for the centre manifold computation and for the first integrals, to estimate the order of the error. The concrete calculations for these cases are left to the reader.

4.8 Invariant tori

Here we note that, using the tools we have developed, it is very easy to compute invariant tori close to any of the libration points of the RTBP. For instance, let us focus on the neighbourhood of the L_5 point of the Earth-Moon RTBP.

$\lambda_0^{(1)}$	$\lambda_0^{(2)}$	n
0.00001	0.00002	4.366
0.00002	0.00004	8.009
0.00004	0.00008	8.011
0.00008	0.00016	8.018
0.00016	0.00032	8.033

Table 3: Estimation of the order of the error.

Figure 1 is a 2-D torus obtained by taking, in the normal form, the actions $I_1 = I_2 = 0.0001$, and $I_3 = 0$. This corresponds to an elliptic (planar) Lyapunov tori obtained from two of the (three) linear oscillations at L_5 (see [30]). Figure 2 corresponds to a 2-D elliptic torus obtained taking $I_1 = I_3 = 0.0001$ and $I_2 = 0$. This torus can also be seen as coming from the linear oscillations around the periodic Lyapunov family associated to the vertical oscillation at L_5 (see [32]). In both cases, we have plotted a dot every 0.1 units of time.

It is not difficult to compute Poincaré sections of these trajectories, to see that they are invariant curves. We left this for the interested reader, as well as the computation of more invariant tori. Finally, let us note that it is also possible to ask for a tori with prefixed frequencies: one has to solve a system of three nonlinear equations to find the corresponding actions. Of course, this is only possible for suitable frequencies.

5 Efficiency Considerations

When one considers the optimality of a given calculation, there are two main things to be taken into account: the algorithm used and its implementation. Here we are not going to discuss the efficiency of the algorithm selected (although there are other possibilities, for example to use quadratic schemes instead of linear ones; see, for instance, [35]), and we are going to focus on their implementation.

Now let us make a few remarks on the optimality of these routines. The implementation we have selected here (to use integer functions –sometimes called "hash functions"– to know the position corresponding to a given exponent and viceversa) allows for very easy implementations, but adds an overhead to the program (the time taken by these functions and the memory used by the integer tables). In some cases, it is possible to use specific orders for the polynomials such that the main operations can be performed directly, without the help of such functions: for instance, when dealing with polynomials of one variable, we can store the coefficient of the monomial x^j into the position number j of the corresponding array, so all the operations can be performed trivially (for instance, the product of two polynomials is `p[i]*q[j]`$\rightarrow$`r[i+j]`). Note that this is still possible in two variables but it becomes more tricky in several variables. Moreover, if the coefficients of the polynomials are sophisticated types (like trigonometric polynomials), the time taken by the hash functions is unnoticeable in front of the time taken by the operation involving

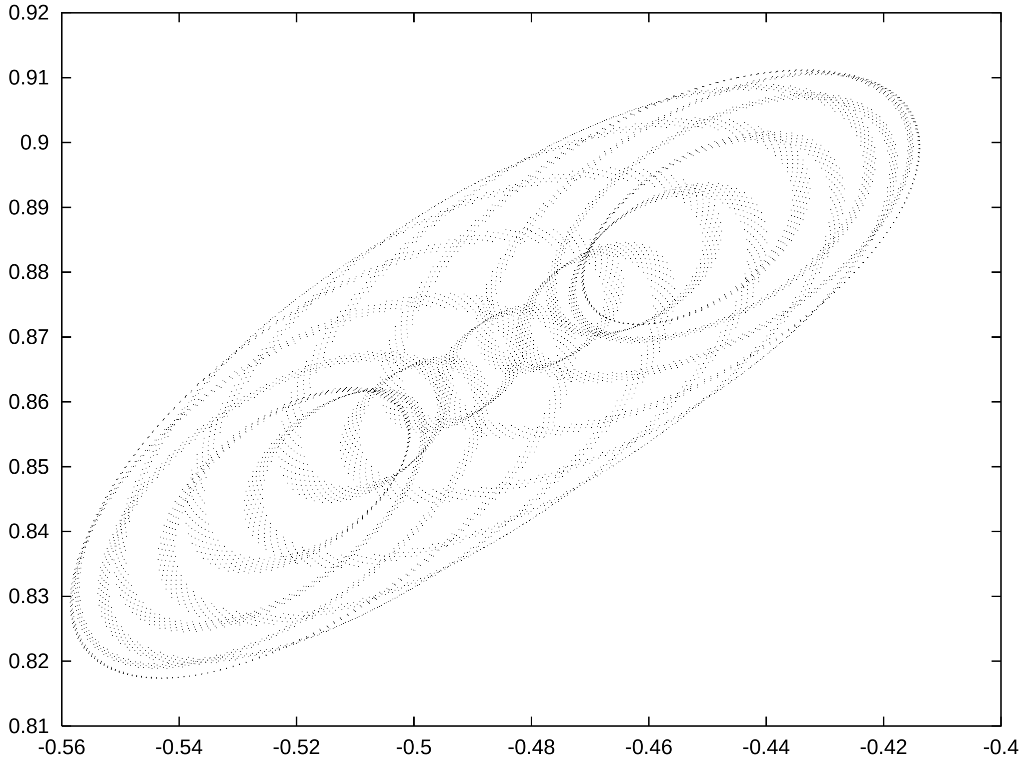

Figure 1: Projection on the (x, y) plane (synodical coordinates) of an elliptic 2-D invariant tori near L_5. The intrinsic frequencies are $\omega_1 = 0.954347344380$ and $\omega_2 = -0.298324062073$. The normal frequency is $\omega_n = 1.00003161731$.

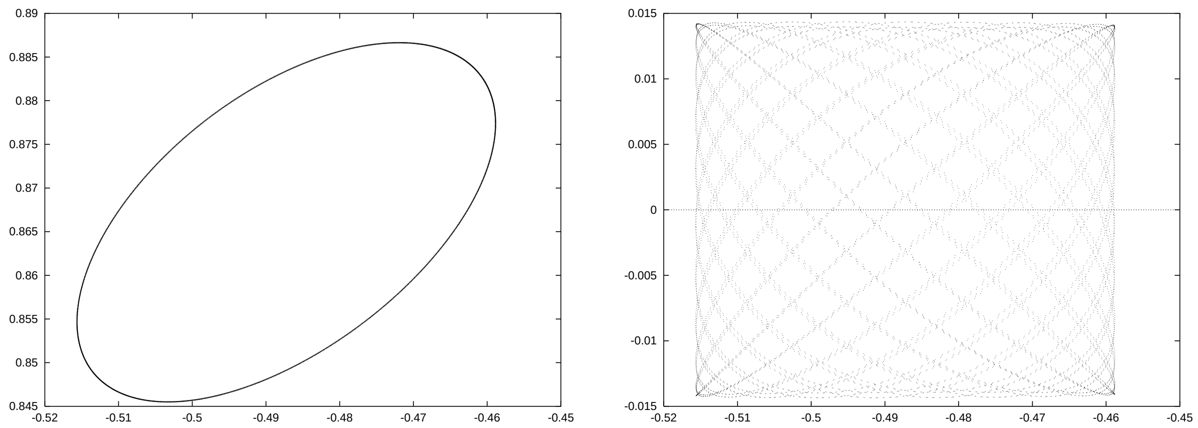

Figure 2: Projections on the (x, y) plane (left) and on the (x, z) plane (right) of a 2-D invariant tori near L_5. The intrinsic frequencies are $\omega_1 = 0.954532905738$ and $\omega_2 = 1.00000846050$. The normal frequency is $\omega_n = -0.298356646196$.

the coefficient. So, in our opinion, the gain obtained by using this kind of tricks is not big enough to compensate for the increase of complexity of the code.

In what follows, the measures of the time needed for the programs to execute have been taken from runs done on a Pentium Pro 200 MHz PC running Linux, with the GNU compiler gcc/g++ version 2.7.2.1. The amount of needed memory has been estimated directly from the size of the expansions.

5.1 Storage

Let us start by considering the efficiency from the point of view of the amount of memory used by the programs. As the memory is allocated and freed dynamically, we will focus on the "worst moment" of the program, that is, when the maximum amount of memory is needed.

Table 4 displays the number of monomials for some of the expansions used here. From this table, and knowing the number of series we use in each program, it is not difficult to have an idea of the order of the amount of memory needed.

5.1.1 Normal forms

On a normal form computation as the one performed here, we use the following expansions (let us denote as n the maximum degree wanted):

1. A power expansion up to degree n of the type `mp6s` (for the Hamiltonian).

2. An auxiliar power expansion (to be used only during the computation of the power expansion of the Hamiltonian) of the same degree as the Hamiltonian.

3. Three polynomials of degree n, of the type `mp6s`, to be used as a working space during the normal form computations.

We have to note that the expansion in item 2 and the three polynomials in item 3 are needed in different places of the program, so we only need to take the maximum of them.

In fact we need a little bit of memory (like the inner tables of the manipulators or the three-variables expansion for the normal form), but the above mentioned series are the most important ones.

Concerning the amount of hard disk memory used, we note that we need

1. A binary file to store the generating function. This is about the size of a power expansion of degree n, of the type `mp6s`.

2. A few extra ASCII files (to store the normal form, the control parameters, etc.) that, as they are very small, we skip them.

Of course, one can modify the program in order to write more information (you can ask for intermediate series) of less (you can skip the writting of the generating function if

mp4s			mp6s			mp6p		
n	$\Phi(n)$	$\sum_{j=0}^{n}\Phi(j)$	n	$\Phi(n)$	$\sum_{j=0}^{n}\Phi(j)$	n	$\Phi(n)$	$\sum_{j=0}^{n}\Phi(j)$
0	1	1	0	1	1	0	0	0
1	2	3	1	4	5	1	2	2
2	6	9	2	13	18	2	8	10
3	10	19	3	32	50	3	24	34
4	19	38	4	70	120	4	56	90
5	28	66	5	136	256	5	116	206
6	44	110	6	246	502	6	216	422
7	60	170	7	416	918	7	376	798
8	85	255	8	671	1589	8	616	1414
9	110	365	9	1036	2625	9	966	2380
10	146	511	10	1547	4172	10	1456	3836
11	182	693	11	2240	6412	11	2128	5964
12	231	924	12	3164	9576	12	3024	8988
13	280	1204	13	4368	13944	13	4200	13188
14	344	1548	14	5916	19860	14	5712	18900
15	408	1956	15	7872	27732	15	7632	26532
16	489	2445	16	10317	38049	16	10032	36564
17	570	3015	17	13332	51381	17	13002	49566
18	670	3685	18	17017	68398	18	16632	66198
19	770	4455	19	21472	89870	19	21032	87230
20	891	5346	20	26818	116688	20	26312	113542
21	1012	6358	21	33176	149864	21	32604	146146
22	1156	7514	22	40690	190554	22	40040	186186
23	1300	8814	23	49504	240058	23	48776	234962
24	1469	10283	24	59787	299845	24	58968	293930
25	1638	11921	25	71708	371553	25	70798	364728
26	1834	13755	26	85463	457016	26	84448	449176
27	2030	15785	27	101248	558264	27	100128	549304
28	2255	18040	28	119288	677552	28	118048	667352
29	2480	20520	29	139808	817360	29	138448	805800
30	2736	23256	30	163064	980424	30	161568	967368
31	2992	26248	31	189312	1169736	31	187680	1155048
32	3281	29529	32	218841	1388577	32	217056	1372104

Table 4: Number of monomials for expansions of the kind `mp4s`, `mp6s` and `mp6p`. Here, n denotes the degree, $\Phi(n)$ is the number of monomials in a polynomial of degree n, and $\sum_{j=0}^{n}\Phi(j)$ is the number of monomials in a expansion up to degree n.

degree	time `nf`	time `cm`	RAM	HD
8	0.40	0.46	0.058	0.025
12	8.01	9.51	0.306	0.153
16	82.25	95.77	1.218	0.609
24	3002.14	3505.71	9.595	4.798
32	48422.61	55769.46	44.435	22.217

Table 5: Time (in seconds) and memory (in megabytes) needed for the normal form (`nf`) and centre manifold (`cm`) computation. We want to note that `cm` needs, to store a temporal file, about the same space as the results. This implies that, to run this program, you need to have twice the column "HD" of disk free.

you are not interested in the change of variables). In such a case, you should re-estimate the amount of memory you need.

In Table 5 we have summarized these estimates on the amount of memory needed. We have assumed that each coefficient is a double precision complex number, that is, each one needs 16 bytes to be stored.

5.1.2 Centre manifolds

The only difference between a normal form and a centre manifold computation (concerning the amount of memory used) appears when realifying the Hamiltonian restricted to the centre manifold. From the program, it is seen that this only affects to the amount of hard disk needed. In Table 5 we have summarized those values. As in the normal form case, we have skipped the size of the ASCII file with the final Hamiltonian, since it is not very big. We note that this file is written after erasing the temporal file, so if it was room for this file, there is enough room for the results. However, if one wants precise estimations of the final amount of used disk, one must take into account the size of that ASCII file. The concrete runs displayed there have been done for the L_1 case of the Earth-Sun system.

5.1.3 First integrals

The calculation of a first integral is a little bit simpler than a normal form one. In fact, the program needs RAM space for the Hamiltonian and the first integral, and disk space for the results as well as a temporary (binary) file used to realify the first integral. In the actual version of the program, the output file is an ASCII file, to be able to look directly at the results using an standard text editor (like `vi` or `emacs`). In Table 6 we have included the time and memory used for several runs of the program. Note that we have been using a lower degree for the calculations. This is because the huge amount of disk space needed to store the output in ASCII format. If one is interested in running to higher orders it should be better to change the program in order to store the first integral in a binary file

degree	time	RAM	HD tmp.	HD final
8	0.38	0.05	0.02	0.11
12	5.36	0.30	0.15	0.67
16	49.98	1.16	0.58	2.66
20	337.09	3.56	1.78	8.17
24	1800.57	9.15	4.58	20.99

Table 6: Time (in seconds) and memory (in megabytes) needed for the calculation of a first integral. The column "HD tmp." only refers to the temporal (binary) files, while the column "HD final" only refers to the final (ASCII) file.

(this is, in fact, very easy using the routines provided here). Then, the amount of disk space is similar to the one used by the centre manifold program (see Table 5).

5.1.4 Changes of variables

Let us discuss the calculations needed to obtain the expansions for the changes of variables corresponding to the normal form case. We will only focus on the direct changes, since the inverse ones need (approximately) the same amount of memory and time.

As before, n will denote the degree of the expansion of the transformation. During the computation of the direct change, we use one expansion up to degree n and three homogeneous polynomials of degree n. In fact, we need polynomials of the type `mp6s` for the transformation corresponding to the four first variables, and of the type `mp6p` for the last two. As the polynomials of the type `mp6s` contain more monomials than the corresponding ones of the type `mp6p`, we have done the memory estimations for the type `mp6s`. They are summarized in Table 7.

A special case is the computation of the changes of variables corresponding to the reduction to the centre manifold. In this case, we obtain six series, each one depending on four variables (see Section 4.4), so the final amount of disk space is smaller than in the normal form case. To estimate the maximum amount of disk space needed during the execution, we note that this occurs during the realification of the last couple of variables. At this moment, we have four real series (of the type `mp4s`) written in the disk, and we write a temporal file with a complex series (of the type `mp6p`) corresponding to the last couple of variables. From these observations, it is not difficult to derive the figures shown in Table 7.

5.2 Speed

Finally let us discuss the optimality, according to the speed, of these routines. To start the discussion, let us focus on the routine that multiplies homogeneous polynomials (see Section 3): for each couple of monomials that we are multiplying we need to know the

degree	time `cvnf`	time `cvcm`	RAM	HD `cvnf`	HD `cvcm`
8	1.11	1.32	0.06	0.07	0.03
12	24.24	29.36	0.29	0.43	0.17
16	272.46	330.22	1.05	1.72	0.63
20	1942.21	2340.47	3.01	5.30	1.90
24	10395.05	12644.89	7.31	13.64	4.80

Table 7: Time (in seconds) and memory (in megabytes) needed for the computation of the changes of variables for the normal form (`cvnf`) and the centre manifold (`cvcm`).

exponents of them and the position to store the result. As every product is an unavoidable operation (we recall that we are discussing the optimality of the implementation, not of the algorithm), all the overhead of this implementation is due to the routines that look for exponents and positions. In fact, if these routines use zero time, the product would be optimal, since all the time spend by the product would correspond to the unavoidable operations. This is also true for the other routines (Poisson brackets, power expansions, etc.). For this reason we say that the optimality of the package is basically given by the optimality of the routines of the files `mp6s.c`, `mp6p.c`, etc. In order to quantify this, we have done a run of the program `nf` using the profiling facilities of the compiler. The results are shown in Table 8. Note that we must eliminate from this table the time used by routine `mcount`, since it des not belong to our program (it has been introduced by the profiler). Then, it is clear that the time taken by routines `exll6s` and `llex6s` is a little bit less than 50% of the total time taken by the program. This implies that if we were able to optimize these routines in order to reduce the time they take to almost zero, the factor in the total gain in speed would be close to 2 (but not better!). Moreover, Table 8 gives precise information about the routines one must optimize to make the program run faster.

Tables 5, 6 and 7 contain the time for several runs of the software. We stress that those are approximate values: time has been taken from a single run of the program, and the amount of RAM memory needed has been estimated form the size of the several expansions used (you should increase these figures a little bit to obtain the real amount of memory used).

6 Error Control

A very important point is to know the numerical errors introduced in the coefficients when this huge amount of computations is performed. A first (heuristic) indication is given by the size of the imaginary parts of the real normal forms, centre manifolds or first integrals that are not zero due to the roundoff errors. It is very natural to take these values as zero because they must vanish in an exact computation.

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
40.15	51.96	51.96	269	193.16	347.79	papu6s
26.48	86.23	34.27				mcount
26.46	120.47	34.24	84136095	0.00	0.00	exll6s
6.02	128.26	7.79	55490539	0.00	0.00	llex6s
0.58	129.01	0.75	14	53.57	6737.27	traham
0.24	129.32	0.31	66	4.70	11.02	pph6s
0.04	129.37	0.05	14	3.57	3.95	cage
0.01	129.38	0.01	14	0.71	1.10	put0
0.01	129.39	0.01	1	10.00	747.48	exp_l5
0.01	129.40	0.01	1	10.00	54.09	reste
0.01	129.41	0.01	1	10.00	15.34	rnf6s
0.00	129.41	0.00	76062	0.00	0.00	kill_nf
0.00	129.41	0.00	38044	0.00	0.00	check_rlf
0.00	129.41	0.00	38032	0.00	0.00	prxk6s
0.00	129.41	0.00	1474	0.00	0.00	ntph6s
0.00	129.41	0.00	164	0.00	0.00	exll3
0.00	129.41	0.00	164	0.00	0.00	llex3
0.00	129.41	0.00	156	0.00	0.00	prxk3
0.00	129.41	0.00	26	0.00	0.00	ntph3
0.00	129.41	0.00	14	0.00	0.00	wpb6s
0.00	129.41	0.00	5	0.00	0.00	uneix
0.00	129.41	0.00	2	0.00	341.69	exrec
0.00	129.41	0.00	1	0.00	0.00	amp3
0.00	129.41	0.00	1	0.00	0.00	amp6s
0.00	129.41	0.00	1	0.00	0.00	ccvl5
0.00	129.41	0.00	1	0.00	0.00	imp3
0.00	129.41	0.00	1	0.00	0.00	imp6s
0.00	129.41	0.00	1	0.00	95140.00	main
0.00	129.41	0.00	1	0.00	94377.18	nf6s
0.00	129.41	0.00	1	0.00	0.00	wctl5
0.00	129.41	0.00	1	0.00	0.00	wcvl
0.00	129.41	0.00	1	0.00	0.00	wea3

Table 8: Output of the profiler for a run (up to degree 16) of the program `nf`. The first column contains the percentage of the total running time of the program used by this function and the fourth column contains the number of times this function is called. The last column indicates the name of the function. We note that routine `mcount` do not belong to our program but to the profiler.

Note that the testing methods discussed in Section 4.7 provide a rough idea of the global amount of error we have accumulated in the computations. This should be enough if we are only interested in numerical results, since this is typically the kind of output obtained from classical numerical methods (think of the solution of an ode, pde or simply the solution of a linear system). In fact, we are in a better position compared with other numerical procedures, since we have a good checking procedure.

However, if one is interested in these methods to be used in a computer assisted proof, we need a much better mechanism to control the error. This is the reason to introduce the interval arithmetic. In what follows, we are going to focus on a normal form computation, although the same ideas can be extended to the other examples considered here.

6.1 Interval arithmetic

In order to carry exact bounds on the error let us assume that, instead of a floating point number, we have an interval such that it contains the number. To add two intervals, we simply add the lower bounds of the interval *using rounding toward* $-\infty$, and we add the upper bounds *using rounding toward* $+\infty$. In this way we ensure that the result of the addition is contained in the final interval. The same ideas can be used to easily derive the operations -, * and $\div$.

The next step is to code efficiently those routines. Fortunately, most of the actual processors allow to the user to alter the rounding mode, to set a rounding toward $\pm\infty$ or to the nearest (this is the default). To do this, many compilers and/or operating systems have suitable functions in their libraries. Here we have used the corresponding routines of the Linux operating system (with the compiler gcc from Gnu), running on an Intel processor. The main disadvantages of this are that the memory requirements are doubled and the execution time is much bigger. This last inconvenient is due to the architecture of the processors, since when the rounding mode of the processor is changed, the pipeline of the processor is re-started with the corresponding loss of performance.

As we have written all the code in C++, it is very easy to use the capacity of overloading the arithmetic operators to substitute the standard complex arithmetic by our interval arithmetic (you can also use [41] if you want to avoid using C++). Then, it is not difficult to obtain the normal form but, instead of the coefficients, we will obtain intervals containing the exact values. This is what allows to derive computer assisted proofs. See [10] and [36] to see concrete applications of these ideas.

6.2 An example with interval arithmetic

Here we have included the computation of the normal form around L_5 for the RTBP using interval artihmetic. The idea is to give a feeling about how these computations are.

In Table 9 we have included the normal form, using double precision interval arithmetic, around the L_5 point of the RTBP, for the mass parameter corresponding to the Earth-Moon system. We have skipped the imaginary parts because they can be assumed to be zero (this is one advantage of interval computations). It is interesting to compare these results with the ones presented in Table 1.

First of all, note the big size of the intervals, specially for the highest degrees displayed.[5] Of course, this does not prove that coefficients in Table 1 contain big numerical errors, but it suggests that we should check this more carefully. In order to do that, we can use a higher precision arithmetic. In this case, we have taken the standard quadruple precision arithmetic that it is contained in the libraries of many compilers (this concrete computation has been done on a Sun workstation). The results are displayed in Table 10. It is interesting to compare this last table with Table 1: if we take the coefficients in Table 10 as exact, we note that the error in the ones of Table 1 is of the order of the imaginary part. This suggests an heuristic criterion to estimate the accuracy of this computation.

Now, it is clear the amplification of errors that we have in this process. There are two (standard) ways of overcome this phenomenon:

1. Intervalar arithmetic. Note that, although the intervals grow very fast, they are still providing exact bounds for the coefficients, that can be useful in order to derive computer assisted proofs (they are going to be a much sharper bound than any other estimation obtained by analytical methods).

2. Multiple precision arithmetic. This is the "brute force" solution, but it is valid in several cases. The advantages are obvious, but one should note that, when dealing with realistic problems, it is not always a feasible option (for instance, the mass parameter corresponding to the Earth-Moon case is only known up to 10 or 11 digits, so there is no gain in using multiple precision).

Of course, in academic problems it is always possible to use a combination of both, to derive very accurate coefficients and/or very sharp estimates for them.

Concerning the normal form around L_5 of the RTBP, let us add that the amplification of errors is bigger when the mass ratio μ is smaller.

Finally, let us note that the routines for interval arithmetic and the extension for quadruple precision are not included in the software.

7 Extensions

In this package we have only considered the case of autonomous Hamiltonians with three degrees of freedom. It is not difficult to extend the ideas and the routines presented here to more degrees of freedom. For instance, to work with a four degrees of freedom Hamiltonian system (without any symmetry) one only needs to write the basic routines of the corresponding file `mp8.c`, and to introduce minor modifications in the other routines.

If one is interested in the computation of normal forms around another objects, in [32] it is explained (from a numerical point of view) the computation of the normal form

[5] We have not tried to optimize the algorithm to minimize the growing of the intervals. It is possible, then, to obtain narrower intervals with a different implementation.

			lower bound	upper bound
1	0	0	9.5450087346978552e-01	9.5450087346991741e-01
0	1	0	-2.9820811951634596e-01	-2.9820811951573489e-01
0	0	1	1.0000000000000000e+00	1.0000000000000000e+00
2	0	0	1.1568661303889360e-01	1.1568661401345537e-01
1	1	0	-1.7127952451731403e+00	-1.7127952303486182e+00
0	2	0	3.3855424323176919e-01	3.3855425662676453e-01
1	0	1	8.9130919836368838e-02	8.9130920112820977e-02
0	1	1	2.2531870640182916e-01	2.2531870757604811e-01
0	0	2	-2.2354591590257877e-03	-2.2354591074729147e-03
3	0	0	-2.9479121860441637e-01	-2.9478447589701773e-01
2	1	0	8.1656201621290165e+00	8.1657691558011720e+00
1	2	0	-5.4586913901624575e+02	-5.4586860598896601e+02
0	3	0	-5.1021371160130911e+01	-5.1021185629532283e+01
2	0	1	-4.3799836956028315e-01	-4.3799552187429924e-01
1	1	1	1.4116969490546651e+01	1.4116998940124972e+01
0	2	1	2.0186927381142823e+00	2.0187190572228246e+00
1	0	2	-5.5905224456048508e-02	-5.5904854484518651e-02
0	1	2	-1.7898271680742539e-01	-1.7898147963031263e-01
0	0	3	-5.1334316020434586e-05	-5.1317165340935330e-05
4	0	0	1.2677680341002997e+00	1.2873345241823699e+00
3	1	0	-3.5434024811722338e+01	-3.4703682770952582e+01
2	2	0	-5.4877274309542030e+04	-5.4872743283411488e+04
1	3	0	3.2220252371445298e+04	3.2226686164319515e+04
0	4	0	3.5177942440398037e+03	3.5192072384618223e+03
3	0	1	2.1707021092443028e+00	2.1811671985342400e+00
2	1	1	1.9986363951466046e+01	2.0216307091992348e+01
1	2	1	1.3647290105217136e+04	1.3647973048501415e+04
0	3	1	1.4506020027436316e+03	1.4508753202967346e+03
2	0	2	2.1927585054381780e+00	2.1948837131021719e+00
1	1	2	-4.9551211330863225e+01	-4.9529208559599283e+01
0	2	2	-1.0188391081203008e+01	-1.0169093839605921e+01
1	0	3	3.5386579632358917e-02	3.5564130055718124e-02
0	1	3	7.0933774363425073e-02	7.1488715816371950e-02
0	0	4	5.1925348264703075e-04	5.2452355219756441e-04

Table 9: Coefficients of the normal form, for the Earth-Moon case, obtained using intervalar arithmetic. Only the real parts are presented.

			real part	imaginary part
1	0	0	0.9545008734698507e+00	0.0000000000000000e+00
0	1	0	-0.2982081195160388e+00	0.0000000000000000e+00
0	0	1	0.1000000000000000e+01	0.0000000000000000e+00
2	0	0	0.1156866135262217e+00	-0.1927100002836750e-32
1	1	0	-0.1712795237759768e+01	-0.8974646880952045e-32
0	2	0	0.3385542499303071e+00	-0.4812484744069060e-32
1	0	1	0.8913091997461692e-01	-0.4814824860968090e-33
0	1	1	0.2253187069890425e+00	-0.1155557966632342e-32
0	0	2	-0.2235459133244455e-02	0.0000000000000000e+00
3	0	0	-0.2947878472529007e+00	-0.1521362462732897e-29
2	1	0	0.8165694658984183e+01	0.1185016987263866e-28
1	2	0	-0.5458688725020474e+03	-0.1174332188770398e-28
0	3	0	-0.5102127839458834e+02	-0.1863024703269571e-28
2	0	1	-0.4379969457189379e+00	-0.1484998366081301e-30
1	1	1	0.1411698421534677e+02	0.3358157455523530e-29
0	2	1	0.2018705897693666e+01	-0.3811527650397076e-30
1	0	2	-0.5590503947042638e-01	0.1150165425481089e-30
0	1	2	-0.1789820982175625e+00	-0.5503192785483820e-31
0	0	3	-0.5132574067108261e-04	-0.1954016422742883e-32
4	0	0	0.1277551279966923e+01	0.6516229084136752e-27
3	1	0	-0.3506885376119049e+02	0.1930958293998747e-25
2	2	0	-0.5487500879622420e+05	0.1108578486500471e-24
1	3	0	0.3222346926821930e+05	0.1264834400557989e-24
0	4	0	0.3518500741321633e+04	-0.2977673781142404e-25
3	0	1	0.2175934654360213e+01	0.1498922726564656e-27
2	1	1	0.2010133553242287e+02	0.4112326735122740e-26
1	2	1	0.1364763157688629e+05	0.4301991684680189e-26
0	3	1	0.1450738661531158e+04	0.6827018990166253e-27
2	0	2	0.2193821109377304e+01	0.2410685221214210e-28
1	1	2	-0.4954020994411146e+02	0.1635601948530436e-27
0	2	2	-0.1017874245948335e+02	0.1985868172512715e-27
1	0	3	0.3547535485371232e-01	-0.4504667333242595e-30
0	1	3	0.7121124512960337e-01	0.9633349924466193e-29
0	0	4	0.5218885184995916e-03	0.1527174289047186e-30

Table 10: Coefficients of the normal form, for the Earth-Moon case, obtained using quadruple precision. The last column contains the imaginary parts of the coefficients, which should be zero.

around a periodic orbit of the spatial RTBP. The routines used there are based in the methodology explained here.

The case in which the Hamiltonian depends on time can also be considered. For instance, let us consider the Hamiltonian of the RTBP with a perturbation that depends periodically on time. In this case, one can still use the routines here but one has to change the basic arithmetic: now, the coefficients of the monomials are going to be Fourier series. We can store Fourier series in complex form as polynomials of one variable, using an array to put the coeficients and using the place inside the array to know the corresponding exponent (in this case one should say frequency instead of exponent). As the relation between positions and frequencies is very easy and one does not need to write any special function for this.[6] Then, one needs to write the arithmetic routines (sums and products) for these Fourier series and to use them instead of the complex arithmetic for the coefficients. Note that this can be easily done if you are using a C extension allowing for overload of arithmetic operators, as C++ or SCC (see [41]) do. Finally, you have to modify the input/output routines accordingly. This is what we have done in [28] or [50], for the case of a periodically perturbed Hamiltonian system.

8 Acknowledgements

The techniques exposed here have been learned or developed with some colleages from the Dynamical Systems Group at Barcelona (`http://www.maia.ub.es/dsg`). I am specially indebted with Carles Simó for the fruitful discussions we have had about this subject. In fact, my first algebraic manipulator (see [12]) started as an undergraduate project at the University of Barcelona.

I want to thank R. de la Llave for suggesting this work, J. Villanueva for his remarks and R. Broucke, A. Giorgilli and J. Henrard for some bibliographical comments. I also want to acknowledge the hospitality of TICAM (University of Texas at Austin), where this project started.

The financial support comes from the Spanish grant DGICYT PB94–0215, the EC grant ERBCHRXCT940460, and the Catalan grant CIRIT 1996SGR–00105. This research was also supported in part by the Institute for Mathematics and its Applications (IMA), with funds provided by the National Science Foundation.

A Basics on Hamiltonian Mechanics

In this appendix we give the basic definitions and properties related to Hamiltonian systems. The information presented here is biased towards the items needed in this

[6] This changes drastically when one has to deal with quasiperiodic time-dependent functions, because the mapping between postions and frequencies is more complex. The main problem comes from the fact that these series are usually a little bit "sparse" and it is very convenient to store only the meaningful coefficients, to save memory. This is used and discussed in [15] and [18].

paper. A more complete and rigorous presentation can be found in any textbook on this subject (see, for instance, [2], [38] or, for a more formal approach, [1]).

To simplify the discussion, from now on we will assume (without explicit mention) that all the functions that will appear here are analytic.

A.1 Basic definitions

A Hamiltonian system is a (continuous) dynamical system whose flow satisfies an ordinary differential equation of the kind:

$$\dot{q} = \frac{\partial H}{\partial p}, \qquad \dot{p} = -\frac{\partial H}{\partial q}. \tag{16}$$

Variable $p \in \mathbb{R}^\ell$ is called momentum and variable $q \in \mathbb{R}^\ell$ is called position. The function $H \equiv H(p, q, t)$ is called the Hamiltonian of the system (16), and equations (16) are known as the Hamilton equations. Moreover, ℓ is known as the number of degrees of freedom of the Hamiltonian H.

If we define the matrix J as:

$$J = \begin{pmatrix} 0 & I \\ -I & 0 \end{pmatrix},$$

where I is the identity matrix $\ell \times \ell$, then we can write equations (16) as:

$$\dot{z} = J\nabla H(z), \quad z = (q, p).$$

As J satisfies $J^\top = -J$, it defines a symplectic form[7] ω^0 on $\mathbb{R}^{2\ell}$:

$$\omega^0(u, v) = u^\top J v, \quad u, v \in \mathbb{R}^{2\ell}.$$

A matrix M is said to be symplectic if it satisfies

$$M^\top J M = J.$$

A function f,

$$\begin{array}{rcl} f : \mathbb{R}^\ell \times \mathbb{R}^\ell & \to & \mathbb{R} \\ (p,\, q) & \mapsto & f(p, q) \end{array}$$

is said to be a first integral of the Hamiltonian H if its surface levels are invariant by the flow (16), this is, if f takes a constant value on each orbit of the system. It is immediate to check that the function H is always a first integral of the Hamiltonian H.

The Poisson bracket of two functions $f(p, q)$ and $g(p, q)$ is defined as:

$$\{f, g\} = \nabla f^\top J \nabla g = \frac{\partial f}{\partial q}\frac{\partial g}{\partial p} - \frac{\partial f}{\partial p}\frac{\partial g}{\partial q}.$$

[7] A symplectic form is a non degenerate bilinear skew symmetric form

It is not difficult to show that, if f is a first integral of the Hamiltonian H, then it must satisfy $\{H, f\} = 0$.

Two functions $f(p, q)$ and $g(p, q)$ are said to be in involution if their Poisson bracket is zero,

$$\{f, g\} = 0.$$

The functions $\{f_j\}_{1\leq j\leq n}$ are said to be independent on some open domain D if the vectors $\{\nabla f_j\}_{1\leq j\leq n}$, defined on the domain D, are linearly independent on each point of the domain.

In the next sections we will use the following property of the Poisson bracket: if P_r and Q_s are homogeneous polynomials of degree r and s respectively, then $\{P_r, Q_s\}$ is an homogeneous plynomial of degree $r + s - 2$.

In what follows, we will assume that all the Hamiltonians that will appear here are autonomous (they do not depend on time) and with ℓ degrees of freedom.

A.2 Basic properties

Let us assume that a Hamiltonian system H has ℓ first independent integrals, $\{f_j\}_{1\leq j\leq \ell}$, that are in involution. Let us define $\mathcal{M}_0$ as

$$\mathcal{M}_0 = \{(p, q) : f_j(p, q) = f_j^{(0)},\ j = 1, \dots, \ell\}.$$

Then, the well-known Liouville-Arnol'd theorem (see [2] or [5]) says that:

1. The manifold $\mathcal{M}_0$ is invariant by the flow.

2. If $\mathcal{M}_0$ is a compact connected manifold,[8] then it is diffeomorphic to the ℓ dimensional torus

$$\mathbb{T}^\ell = \{(\phi_1, \dots, \phi_\ell) \mathrm{mod}\ 2\pi\}.$$

In this last case it is possible to introduce, by means of a change of variables $(p, q) = F(I, \phi)$ ($I \in \mathbb{R}^\ell$ is the new momentum and $\phi \in \mathbb{T}^\ell$ is the new position) the so-called action-angle variables (I are the actions and ϕ are the angles). In these variables the Hamiltonian does not depend on the angles, $H = H(I)$, so the equations of motion are of the form

$$\dot{I} = 0, \qquad \dot{\phi} = \frac{\partial H}{\partial I} \equiv \omega(I).$$

Note that these equations can be easily integrated:

$$I(t) = I_0, \qquad \phi(t) = \omega(I_0)t + \phi_0.$$

[8]Of course, there are other possibilities that we will not discuss here, since they will not be necessary in this presentation.

If the values $\omega(I_0) \equiv \omega_0$ are linearly independent over the rationals, each solution is a dense quasiperiodic trajectory on a torus of dimension ℓ. It is very common to use the frequency vector to identify a concrete torus of the system. If the map

$$I \mapsto \frac{\partial H}{\partial I}(I) \equiv \omega(I)$$

is a diffeomorphism (between suitable domains), it is also possible to identify a torus by the value of the action variable.

If $\langle k, \omega_0 \rangle = 0$ for some $k \in \mathbb{Z}^\ell$, then the orbits on this torus are not dense: if there are ℓ_i independent frequencies, the torus $I = I_0$ contains a $(\ell - \ell_i)$-parametric family of ℓ_i dimensional tori, being each one densely filled by any trajectory starting on it. These tori of dimension ℓ_i are known as lower dimensional tori, while the tori of dimension ℓ are called maximal dimensional ones.

A.3 Canonical transformations

Now let us consider the effect that the changes of variables have on Hamiltonian systems. Let $H(q, p)$ be a Hamiltonian function, and let us consider a change of variables $(q, p) = \Psi(x, y)$. Note that the Hamilton equations obtained from the Hamiltonian $H \circ \Psi$ can be different from the equations obtained applying the transformation Ψ to the Hamilton equations related to H. When these differential equations coincide, it is said that the transformation Ψ preserves the Hamiltonian form.

A change of variables is called canonical when it preserves the Hamiltonian form (for any Hamiltonian function). It is not difficult to show a transformation is canonical if and only if the differential of the change (on any point) is a symplectic matrix.

Canonical transformations are very useful both from the theoretical and the practical points of view, since they allow to work on a single function (the Hamiltonian) instead of a system of 2ℓ differential equations.

Note that to produce canonical changes of variables is not an easy problem, since it is very difficult to impose that the differential be a symplectic matrix. Fortunately, there exists several techniques to produce such transformations. The one that we will use here is based on the following properties of the Hamiltonian flows:

1. Let $\Phi_t(x, y)$ be the time t flow of a Hamiltonian system. Then, $(q, p) = \Phi_t(x, y)$ is a canonical transformation.

2. Let $G(q, p)$ a Hamiltonian system with ℓ degrees of freedom, and let $(q_0(t), p_0(t))$ be a solution of G. Then,

$$\frac{d}{dt} f(q_0(t), p_0(t)) = \{f, G\}\,(q_0(t), p_0(t)), \tag{17}$$

for any smooth function f.

Now, it is not difficult to see that to transform a Hamiltonian H by means of the time 1 flow of a Hamiltonian G, we can apply the formula

$$\hat{H} \equiv H + \{H, G\} + \frac{1}{2!}\{\{H, G\}, G\} + \frac{1}{3!}\{\{\{H, G\}, G\}, G\} + \cdots, \tag{18}$$

where $\hat{H}$ denotes the transformed Hamiltonian. This formula is deduced applying the Taylor formula for the transformation and using (17) for the derivatives involved. The Hamiltonian G is usually called the generating function of the change of variables.

The expression (18) is very suitable for effective computations, since it can be easily implemented on a computer. Note that all the operations involved are very simple if we are working with some kind of expansions (power expansions, Fourier expansions, etc.). One can argue that the problem for this kind of transformation (for a practical point of view) is that it is defined by an infinite series. This is not a problem since we usually work with a finite truncation of these series. This will produce a high order approximation to the results wanted that, in many cases, are good enough for pratical purposes. On the other hand, it is possible to derive rigorous estimates on the size of this remainder so one can obtain bounds on the error of the results obtained with the truncated series (see [43], [28] or [32] for numerical examples of this).

A.4 Normal forms

We are going to restrict ourselves to the normal form around a fixed point of a Hamiltonian system. For normal forms around more complex objects (like periodic orbits or invariant tori), see [9], [30] or [31].

Let H be a real analytic Hamiltonian of ℓ degrees of freedom having an elliptic equilibrium point that, without loss of generality, we can assume that it is located at the origin. The case in which the equilibrium is of the type "some centres" times "some saddles" will be discussed later.

Let us start by expanding H in power series around the origin,

$$H(q, p) = H_2(q, p) + H_3(q, p) + H_4(q, p) + \cdots, \tag{19}$$

where $H_j(q, p)$ is an homogeneous polynomial of degree j in the variables (q, p). Our purpose is to perform (canonical) transformations in order to simplify as much as possible this expansion. Ideally, one would like to remove completely all the H_j with $j \geq 3$ but we will see that this is, generically, impossible. What we will show here is that, under some hypotheses, it is possible to remove the necessary terms to produce integrable approximations to the dynamics.

In order to simplify the subsequent steps, it is very convenient to simplify $H_2(q, p)$. Let A be the linearization of the Hamiltonian flow of H around the origin (i.e., $A = J\nabla H_2(0, 0)$). As A is an elliptic matrix, we can reduced it to $\hat{A} = C^{-1}AC$, being C a real matrix and $\hat{A}$ of the following form

$$\hat{A} = \begin{pmatrix} \hat{A}_1 & & \\ & \ddots & \\ & & \hat{A}_\ell \end{pmatrix},$$

where the elements outside the $\hat{A}_j$ are zero, and

$$\hat{A}_j = \begin{pmatrix} 0 & \omega_j \\ -\omega_j & 0 \end{pmatrix}, \quad \omega_j \in \mathbb{R}, \quad j = 1, \dots, \ell.$$

It is not difficult to check that this change can be selected canonical. If we call (x, y) to the new variables (x is the position and y the momentum), we want to note that the order ("permutation") of these variables to achieve this form for $\hat{A}$ is $(x_1, y_1, x_2, y_2, \dots, x_\ell, y_\ell)$. In these coordinates, H_2 takes the form

$$\hat{H}_2(x, y) = \sum_{j=1}^{\ell} \frac{\omega_j}{2} \left(x_j^2 + y_j^2 \right). \tag{20}$$

In order to simplify the computations in the normal form process (basically, the computations of generating functions), we will perform the following (linear and symplectic) transformation:

$$x_j = \frac{q_j + \sqrt{-1} p_j}{\sqrt{2}}, \quad y_j = \frac{\sqrt{-1} q_j + p_j}{\sqrt{2}}, \tag{21}$$

where we call (again) (q, p) to the new variables. In these variables, H_2 takes the form

$$H_2(q, p) = \sum_{j=1}^{\ell} \sqrt{-1} \omega_j q_j p_j.$$

In what follows, we will denote $\omega = (\omega_1, \dots, \omega_\ell)$, and we will assume that the values ω_j, $1 \leq j \leq \ell$, are linearly independent over the rationals.

Let us assume that the initial expansion (19) has been rewritten in these variables, and we want to apply a sequence of canonical transformations (based on the scheme (18)). Let us start by trying to remove H_3, by means of a generating function G_3 that is also a homogeneous polynomial of degree 3. From (18) it is immediate to see that the monomials of degree 3 of the transformed Hamiltonian $\hat{H}$ obtained using a generating function G_3 are given by

$$\hat{H}_3 = H_3 + \{H_2, G_3\}.$$

Let us try to select a G_3 such that $\hat{H}_3$ is zero. To this end, we introduce the following notation: if $z = (z_1, \dots, z_n)$ and $k = (k_1, \dots, k_n) \in \mathbb{N}^n$, we define

$$z^k = z_1^{k_1} \cdots z_n^{k_n}, \quad |k| = k_1 + \cdots + k_n.$$

Then, we write H_3 and G_3 as

$$H_3(q,p) = \sum_{|k_q|+|k_p|=3} h_{k_q,k_p} q^{k_q} p^{k_p}, \quad G_3(q,p) = \sum_{|k_q|+|k_p|=3} g_{k_q,k_p} q^{k_q} p^{k_p}.$$

Next step is to solve the equation $\hat{H}_3 = 0$. Note that $L_{H_2}(\cdot) = \{H_2, \cdot\}$ is a linear operator in diagonal form, because

$$L_{H_2}(q^{k_q} p^{k_p}) = \left\{ H_2, q^{k_q} p^{k_p} \right\} = \sqrt{-1}\,\langle k_p - k_q, \omega\rangle\, q^{k_q} p^{k_p}.$$

Note that this diagonal form is due to the complex coordinates introduced in (21). Now it is very easy to find a G_3 such that $\{H_2, G_3\} = -H_3$:

$$G_3(q,p) = \sum_{|k_q|+|k_p|=3} \frac{-h_{k_q,k_p}}{\sqrt{-1}\,\langle k_p - k_q, \omega\rangle} q^{k_q} p^{k_p}.$$

Of course, we need that the denominators $\langle k, \omega\rangle$ do not vanish for any $k \in \mathbb{Z}^\ell \setminus \{0\}$. As $|k_q| + |k_p| = 3$, this condition is automatically satisfied if the components of the frequency vector $\omega = (\omega_1, \ldots, \omega_\ell)$ are linearly independent over the rationals.

We rename the transformed Hamiltonian as H, that now takes the form

$$H(q,p) = H_2(q,p) + H_4(q,p) + H_5(q,p) + \cdots.$$

The next step is to look for a generating transformation G_4 (a homogeneous polynomial of degree 4), to remove the monomials of degree 4 from H. Note that this is not possible in general, since L_{H_2} has some zero eigenvalues:

$$L_{H_2}(q^k p^k) = \left\{ H_2, q^k p^k \right\} = 0.$$

Note that this never happens for monomials of odd degree. The monomials of the type $q^k p^k$ are usually called resonant monomials or unavoidable resonances. Hence, when we try to solve the equation $L_{H_2}(G_4) = -H_4$ we only can solve for the monomials of H_4 of the form $q^{k_q} p^{k_p}$, with $k_q \neq k_p$:

$$G_4(q,p) = \sum_{\substack{|k_q|+|k_p|=4 \\ k_q \neq k_p}} \frac{-h_{k_q,k_p}}{\sqrt{-1}\,\langle k_p - k_q, \omega\rangle} q^{k_q} p^{k_p}.$$

With this change, H takes the form (we call again H to the transformed Hamiltonian)

$$H(q,p) = H_2(q,p) + \bar{H}_4(q,p) + H_5(q,p) + \cdots, \tag{22}$$

where

$$\bar{H}_4 = \sum_{|k|=2} \bar{h}_k q^k p^k.$$

Fortunately, the monomials present in $\bar{H}_4$ do not obstruct integrability: let us skip the terms in (22) of order bigger than 4 (this is what we call the normal form of the initial Hamiltonian (19) up to degree 4). Let us apply the canonical transformation

$$x_j = I_j^{1/2} \exp(\sqrt{-1}\phi_j), \quad y_j = -\sqrt{-1} I_j^{1/2} \exp(-\sqrt{-1}\phi_j), \quad j = 1, \ldots, \ell. \tag{23}$$

so that the truncated Hamiltonian takes the form

$$H = H(I) = \langle \omega, I \rangle + H_2(I), \quad H_2(I) = \sum_{|k|=2} \bar{h}_k I^k, \quad I = (I_1, \ldots, I_\ell).$$

This is now an integrable Hamiltonian, that gives an approximate description of the dynamics around the equilibrium point. The equations of motion are

$$\dot{I} = 0, \quad \dot{\phi} = \frac{\partial H(I)}{\partial I} = \bar{\omega}(I).$$

The solutions are $I = I_0$ (that correspond to invariant tori) and $\phi = \bar{\omega}(I_0)t + \phi_0$, that is a quasiperiodic flow on the torus.

Of course, the process of reduction to normal form can be done up to any finite order. Skipping the remainder and using (23) we obtain a Hamiltonian like

$$H = H(I) = \langle \omega, I \rangle + \sum_{n=2}^{N} H_n(I).$$

A.4.1 On the convergence

Generically, the normal form reduction is a divergent process. The divergence is mainly due to the effect of the divisors $\langle k, \omega \rangle$ that appear in the generating functions (in fact, it is possible to have divergence even in the absence of small divisors, see [25]). In order to control the size of these denominators, it is usual to ask for a Diophantine condition like

$$|\langle k, \omega \rangle| > \frac{c}{|k|^\gamma}, \quad k \in \mathbb{Z} \setminus \{0\}, \quad \gamma > \ell - 1. \tag{24}$$

This allows to derive estimates on the size of the remainder obtained when we stop the normal form to some order N. Note that the set of ω such that condition (24) is not satisfied has Lebesgue measure $O(c)$.

In fact, one may look at a normal form as a power expansion of a non-analytic C^∞ function at the origin. The power series is divergent but, if we stop the expansion to some order N, the remainder behaves like $O(R^{N+1})$, where R denotes the distance to the origin. This last property is what makes these expansions useful.

A.5 Stability

Here we will explain some of the applications of the normal forms. Let us consider the neighbourhood of an elliptic equilibrium point (that we locate at the origin) of a $\ell > 1$

degrees of freedom autonomous Hamiltonian system $H(q,p)$. Consider an initial condition close to the origin. We are interested in knowing if the corresponding trajectory will be close to the origin for all times (stability in the sense of Lyapounov), or if it is going to escape to a distance $O(1)$ from the equilibrium point.

A.5.1 The Dirichlet theorem

This is a particular case in which the stability problem can be easily solved. Let us call M to the Hessian matrix of the Hamiltonian at the origin (we recall that M is symmetric and that $\nabla_{(q,p)}H(0,0) = 0$). Assume that M is a positive definite matrix. Then, the Dirichlet theorem says that origin is Lyapounov stable.

The proof is based on the fact that, close to the point, the level surfaces of the Hamiltonian are "like ellipsoids" having the origin inside (those manifolds are of codimension 1 so they split the phase space). Then, as they are invariant for the dynamics, they act as a barrier that the trajectories starting near the point can not cross. Note that the same argument holds if there exists a first integral, defined on a neighbourhood of the origin, that is positive definite at $(0,0)$.

Unfortunately, there are many interesting cases where the matrix M is not positive definite and, hence, we need a different kind of results to study the stability.

A.5.2 KAM and Nekhoroshev theory

In the last section we have seen that, using a finite number of steps of a normal form scheme, we can put the Hamiltonian into the form

$$H(x,y) = H_2(x,y) + \sum_{j=3}^{2N} \bar{H}_j(x,y) + \mathcal{R}_{2N}(x,y).$$

Now, using condition (24) it is possible to derive estimates on the size of the remainder $\mathcal{R}_{2N}$ that are of the kind $c_1 \exp(-c_2(1/R)^{2/(\gamma+1)})$ ($c_1 > 0$, $c_2 > 0$). Here R denotes the radius of the ball centreed at the origin on which we take the norm of $\mathcal{R}_{2N}$, and it is assumed to be sufficiently small. This has been obtained optimizing the size of the remainder with respect to the degree up to which the normal form is obtained, for each value of R.

From this bound on the remainder, it is not diffcult to obtain lower bounds on the diffusion time (i.e., the time to move away) around the point. For instance, if we call $T(R)$ to the time to go out from a ball of radius $2R$ starting in a ball of radius R, we have

$$T(R) \geq c_3 \exp\left(c_4 \left(\frac{1}{R}\right)^{\frac{2}{\gamma+1}}\right),$$

being c_3 and c_4 positive constants. Of course, this is not a proof of stability but a "bound on the unstability". This kind of estimates are what is usually called Nekhoroshev estimates.

A second approach is to try to remove completely the remainder. This can not be done using the normal form scheme we have explained in the previous sections, but it can be done through a Newton method. This is a quadratically convergent iterative scheme, that only converges on a Cantor set of the phase space. On this Cantor set, the trajectories take place on invariant tori and, hence, they never go away from a vicinity of the point. The Lebesgue measure of the complementary of this Cantor set can be bounded by a quantity like $c_5 \exp(-c_6(1/R)^{2/(\gamma+1)})$, $c_5 > 0$, $c_6 > 0$. This kind of results belong to the so-called KAM theory. To decide about the stability we must take into account the motion outside the Cantor set of invariant tori. For instance, let us consider first the case $\ell = 2$. The phase space is four dimensional and, fixing the energy level $H = h$ we restrict to a three dimensional space. The invariant tori are of dimension 2 so they split the phase space and, hence, this allows to conclude the Lyapounov stability of the elliptic point. The case $\ell = 3$ (or bigger) is much more difficult. The reason is the following: fixing the energy level produces a five dimensional invariant manifold and the invariant tori are three dimensional so they do not split phase space and we can not conclude stability. In fact, the stability of Hamiltonian systems with three or more degrees of freedom is today an open question. The more accepted conjecture says that they are, generically, unstable (see [3]). The unstability mechanism is usually known as Arnol'd diffusion.

It is outside the scope of this paper to give detailed explanations of these results. We refer to books like [4] or [5] for a general explanation, and to [30] for more concrete results around invariant objects (like elliptic points, periodic orbits or invariant tori).

A.6 Centre manifolds

Let us consider now a Hamiltonian with three degrees of freedom, in a neighbourhood of an equilibrium point of the type centre$\times$centre$\times$saddle, that we will assume to be the origin. Of course, this is an unstable equilibrium point but we are interested in the existence of trajectories that remain close to the point for all times. If we consider the linearization of the vectorfield at this point, and we skip the hyperbolic part, we obtain a couple of harmonic oscillators. Hence, for the linearized vectorfield, we have a couple of families of periodic orbits near the point, plus the quasiperiodic solutions obtained as product of the two families of periodic orbits. These quasiperiodic solutions are sometimes called Lissajous orbits. Let us consider now the effect of the nonlinear terms of the vectorfield on these bounded solutions. Under generical conditions the well-known Lyapounov centre theorem says that, for each linear (periodic) oscillation, there exists a one-parametric family of periodic orbits of the complete Hamiltonian system that emanates from the point in a tangent way to the linear family of oscillations. The limit frequency of these periodic orbits at the fixed point is the frequency of the linear oscillations (for a proof see, for instance, [42]). A similar result holds for the Lissajous orbits. Under general hypotheses, it can be shown that these linear oscillations can be extended to the complete system as a Cantorian family of invariant tori. Moreover, the measure of the gaps between tori is exponentially small with the distance to the origin (for the proofs, see [30]).

To give a more accurate description of the dynamics around the point, let us apply a normal form technique, as it has been done in previous sections. We start expanding the Hamiltonian in power series around the point (as in (19)). Next we write the second degree terms H_2 in real coordinates such that

$$H_2(x,y) = \lambda x_1 y_1 + \frac{\omega_2}{2}\left(x_2^2 + y_2^2\right) + \frac{\omega_3}{2}\left(x_3^2 + y_3^2\right), \qquad (\lambda, \omega_2, \omega_3) \in \mathbb{R}^3.$$

The coordinates x_1, y_1 are already in diagonal form, so we only need to complexify the couples (x_2, y_2) and (x_3, y_3). Using the change (21) for these two couples we obtain

$$H_2(q,p) = \lambda q_1 p_1 + \sqrt{-1}\omega_2 q_2 p_2 + \sqrt{-1}\omega_3 q_3 p_3.$$

Now we can start a normal form process as the one described in Section A.4 but, instead of killing all the possible monomials, we will only kill the monomials such that the exponent of q_1 is different from the exponent of p_1 (for a different killing criterion, see [46]). That is, the generating function used to remove monomials of degree n will be of the form

$$\sum_{k_{q_1} \neq k_{p_1}} \frac{-h_{k_q k_p}}{(k_{p_1} - k_{q_1})\lambda + \sqrt{-1}(k_{p_2} - k_{q_2})\omega_2 + \sqrt{-1}(k_{p_3} - k_{q_3})\omega_3}.$$

As $k_{q_1} \neq k_{p_1}$, the denominators of the generating function are bounded from below and this is the reason for which the normal form process diverges very slowly (like an harmonic series, see [25]).

If we stop this scheme after a finite number of steps, we obtain a Hamiltonian like

$$H(q,p) = H_N(q_1 p_1, q_2, q_3, p_2, p_3) + \mathcal{R}(q,p).$$

Neglecting the remainder $\mathcal{R}$ (it is very small near the origin), we can define $I_1 = q_1 p_1$ (this is a canonical change if we define properly the corresponding angle variable) to obtain a Hamiltonian $H_N(I_1, q_2, q_3, p_2, p_3)$. Note that the equation corresponding to the variable I_1 is $\dot{I}_1 = 0$ so it is a first integral of the system. Selecting the value $I_1 = 0$ we are restricting the Hamiltonian H_N to an invariant manifold that is tangent at the origin with the linear central part of the system. This is the so called reduction to the centre manifold.

Once I_1 has been replaced by 0, we have obtained a two degrees of freedom Hamiltonian system $H_c \equiv H_N(0, q_2, q_3, p_2, p_3)$, where the origin is an elliptic equilibrium point. It is not difficult to produce a qualitative description of the dynamics of H_c: the phase space is four dimensional, so let us fix a energy level $H_c = h_c$ to reduce to a three dimensional phase space. Now, Poincaré sections are two dimensional and can be plotted easily. Doing several plots for several values of h_c one gets a description of the trajectories that remain close to the origin. The dynamics of the initial Hamiltonian near the origin can be obtained adding the hyperbolic part that we have skipped when reducing to the centre manifold. See [26] or [27] for examples of this.

Although this reduction is divergent in general, we can apply KAM techniques to show, under suitable hypotheses, the existence of a Cantorian centre manifold,[9] completely filled up by invariant tori. The complementary of the measure of this manifold (in the parameters space) decreases exponentially with the distance to the origin. See [30] for more details.

[9]This manifold is parametrized by two parameters, and each parameter moves on a Cantor set.

A.7 First integrals

Again, let us consider the dynamics near an equilibrium point of a Hamiltonian system. Now we are interested in producing first integrals of the motion. Of course, if the Hamiltonian is not integrable (this is, in fact, the general case) these integrals are not going to exist but, as we will see, it is still possible to produce approximate first integrals that can be useful for some applications.

To simplify the discussion, we will assume that the equilibrium point is at the origin and that it is of elliptic type. The case in which some directions are hyperbolic can be done in a very similar way.

As in the previous cases, let us assume that the Hamiltonian is expanded in power series (as in (19)), with H_2 in diagonal form (as in (20)). Let us denote by F the (wanted) first integral, that we will expand in power series around the origin as $F = \sum_{j\geq 2} F_j$, where F_j denotes a homogeneous polynomial of degree j. From the condition $\{H, F\} = 0$ it is immediate to obtain the following recurrence:

$$\{H_2, F_n\} = -\sum_{j=3}^{n} \{H_j, F_{n-j+2}\}. \tag{25}$$

Hence, due to the diagonal form of H_2, it is very easy to solve F_n in terms of $F_2, \dots, F_{n-1}$, assuming the standard non resonant conditions on the frequencies of the point.[10] Then, given a F_2, we can compute the following terms F_3, F_4 and so on.

As usual, the series $F = \sum_{j\geq 2} F_j$ is divergent. However, from its asymptotic character we can derive quasi-integrals of motion by simply truncating the series to finite order. This means that, if f_n denotes a quasi-integral and $(q(t), p(t))$ is an orbit of the Hamiltonian system H then,

$$\dot{f}_n(q(t), p(t)) = \{H, f_n\}\,(q(t), p(t))$$

Bounding the Poisson bracket of this formula in a neighbourhood of the elliptic point one can derive estimates on the diffusion time near the point. For an application of these techniques, see [11]. See also [37] for an early construction of quasi-integrals.

B Linear normal form for the equilibrium points of the RTBP

Let us start with a brief description of the so-called Restricted Three Body Problem (RTBP). More details can be obtained in (almost) any textbook on Celestial Mechanics, like [52].

Let us consider two punctual masses (usually called primaries) that attract each other according to the gravitational Newton's law. Let us assume that they are moving in

[10] As it has been mentioned before, the operator $L_{H_2}(\cdot) = \{H_2, \cdot\}$ is not bijective. Then it is possible that, if the right hand side of (25) contains resonant monomials, this equation can not be solved. There are several cases when it can be proved that such monomials never appear. See [11] for a discussion of this.

circular orbits around their common centre of masses, and let us consider the motion of an infinitessimal particle (here, infinitessimal means that its mass is so small that we neglect the effect it has on the motion of the primaries and we only take into account the effect of the primaries on the particle) under the attraction of the two primaries. The study of the motion of the infinitessimal particle is what is known as RTBP.

To simplify the equations of motion, let us take units of mass, length and time such that the sum of masses of the primaries, the gravitational constant and the period of the motion of the primaries is 1, 1 and 2π respectively. With these units the distance between the primaries is also equal to 1. We denote as μ the mass of the smallest primary (the mass of the biggest is then $1-\mu$), $\mu \in (0, \frac{1}{2}]$.

The system of reference is defined as follows: the origin is taken at the centre of masses of the primaries, the X-axis points to the biggest primary (with this orientation), the Z-axis points to the direction of the vector of angular motion of the primaries with respect to their common centre of masses (it is perpendicular to the plane of motion) and the Y-axis is defined such that we obtain an orthogonal, positive-oriented system of reference. Note that we have defined a rotating system of reference, that is usually called synodic. In this system, the primary of mass μ is at the point $(\mu-1,0,0)$ and the one of mass $1-\mu$ is at $(\mu,0,0)$.

Defining momenta as $P_X=\dot{X}-Y$, $P_Y=\dot{Y}+X$ and $P_Z=\dot{Z}$, the equations of motion can be written in Hamiltonian form. The corresponding Hamiltonian function is

$$H=\frac{1}{2}(P_X^2+P_Y^2+P_Z^2)+YP_X-XP_Y-\frac{1-\mu}{r_1}-\frac{\mu}{r_2}, \tag{26}$$

being $r_1^2=(X-\mu)^2+Y^2+Z^2$ and $r_2^2=(X-\mu+1)^2+Y^2+Z^2$.

It is well-known that the system defined by (26) has five equilibrium points. Two of them can be found as the third vertex of the two equilateral triangles that can be formed using the two primaries as vertices (usually called $L_{4,5}$ or Lagrangian points). The other three lay on the X-axis and are usually called $L_{1,2,3}$ or Eulerian points (see Figure 3).

In the next sections we will study the linear behaviour around these five equilibrium points. We will obtain the linear normal form around them as well as the corresponding (symplectic) changes of variables. These calculations are summarized in [14] and [15], and here we give them in detail. They have been included for completeness.

B.1 The equilateral points

The equilibrium points L_4 and L_5 are located at $(\mu-\frac{1}{2}, \mp\frac{\sqrt{3}}{2}, 0)$, where the upper ("$-$") sign is for L_4 while the lower ("+") one is for L_5. These points are known to be linearly stable when the mass parameter μ is less than the Routh critical value $\mu_R=\frac{1}{2}\left(1-\sqrt{\frac{23}{27}}\right)\approx 0.03852$. In what follows we will assume that our mass parameter is less than μ_R (the interested reader should not have any problem to complete the opposite case).

The first step is to translate the origin of coordinates to the equilibrium point. This

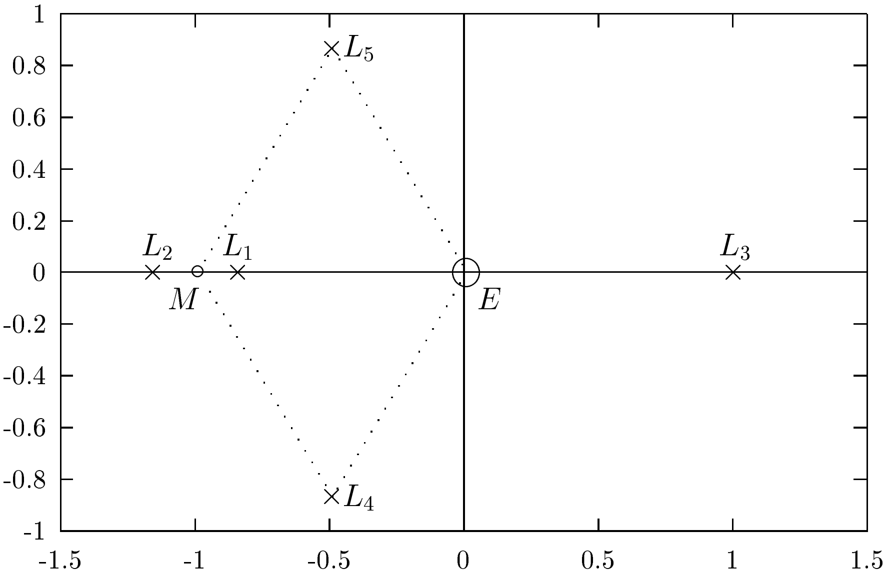

Figure 3: The five equilibrium points of the RTBP. The graphic corresponds to the Earth–Moon case, $\mu \approx 0.01215$.

is done applying the (symplectic) change

$$\begin{array}{rclrcl} X & = & x+\mu-\frac{1}{2}, & P_X & = & p_x \pm \frac{\sqrt{3}}{2}, \\ Y & = & y \mp \frac{\sqrt{3}}{2}, & P_Y & = & p_y + \mu - \frac{1}{2}, \\ Z & = & z, & P_Z & = & z, \end{array}$$

to the Hamiltonian (26). As before, the upper sign is for the L_4 case and the lower one for the L_5 case (this rule for the signs will be used along this section). To simplify notation, we call again H to the Hamiltonian obtained,

$$H = \frac{1}{2}(p_x^2 + p_y^2 + p_z^2) + yp_x - xp_y + \left(\frac{1}{2} - \mu\right) x \mp \frac{\sqrt{3}}{2} y - \frac{1-\mu}{r_{PS}} - \frac{\mu}{r_{PJ}},$$

where $r_{PS}^2 = (x-x_S)^2 + (y-y_S)^2 + z^2$, $r_{PJ}^2 = (x-x_J)^2 + (y-y_J)^2 + z^2$, $x_S = 1/2$, $y_S = \mp\sqrt{3}/2$, $x_J = -1/2$ and $y_J = \mp\sqrt{3}/2$. Note that $(x_S, y_S, 0)$ are the coordinates of the big primary in the new coordinates and that $(x_J, y_J, 0)$ is the position of the small one.[11]

The next step is to expand H around the origin. Note that, as the origin is an equilibrium point, the first order terms must vanish (we simply don't care about the constant value $H(0)$, since it is irrelevant to the dynamics). The first non-trivial terms

[11] The subindices correspond to "Sun" and "Jupiter". They provide a classical example for the RTBP, where the small particle can be an asteroid.

are of second order and they are responsible for the linear dynamics around the point. They are

$$H_2 = \frac{1}{2}(p_x^2 + p_y^2 + p_z^2) + yp_x - xp_y + \frac{1}{8}x^2 - \frac{5}{8}y^2 - axy + \frac{1}{2}z^2,$$

where $a = \pm\frac{3\sqrt{3}}{4}(1-2\mu)$. Note that the behaviour in the (z, p_z) directions is uncoupled of the behaviour in the (x, y, p_x, p_y) directions. Moreover, the motion on the z-axis corresponds to an harmonic oscillator with frequency 1 (for all μ), that it is already in (real) normal form. Hence, we restrict ourselves to the (x, y, p_x, p_y)-plane:

$$H_2 = \frac{1}{2}(p_x^2 + p_y^2) + yp_x - xp_y + \frac{1}{8}x^2 - \frac{5}{8}y^2 - axy. \tag{27}$$

Let us define the 4×4 matrix J as

$$J = \begin{pmatrix} 0 & I_2 \\ -I_2 & 0 \end{pmatrix},$$

where I_2 denote the 2×2 identity matrix. The equations of motion of (27) are given by the linear system

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{p}_x \\ \dot{p}_y \end{pmatrix} = J\nabla H_2 = J\mathrm{Hess}(H_2) \begin{pmatrix} x \\ y \\ p_x \\ p_y \end{pmatrix}. \tag{28}$$

An easy computation shows that the matrix $M = J\mathrm{Hess}(H_2)$ is given by

$$M = \begin{pmatrix} 0 & 1 & 1 & 0 \\ -1 & 0 & 0 & 1 \\ -\frac{1}{4} & a & 0 & 1 \\ a & \frac{5}{4} & -1 & 0 \end{pmatrix}. \tag{29}$$

The characteristic polynomial is $p(\lambda) = \lambda^4 + \lambda^2 + \frac{27}{16} - a^2$. From this expression it is easy to obtain that system (28) is stable if $\mu \le \mu_R = \frac{1}{2}\left(1 - \sqrt{\frac{23}{27}}\right)$ (this is the so-called Routh mass) and unstable if $\mu_r < \mu \le \frac{1}{2}$. As we are studying the case $\mu < \mu_R$, we assume that the solutions of $p(\lambda) = 0$ are all purely imaginary, that is, $\lambda_j = \pm\omega_j\sqrt{-1}$, $j = 1, 2$. The real values ω_j are the frequencies of the linear oscillations at the equilibrium points $L_{4,5}$, and it is trivial to show that they always differ when $0 < \mu < \mu_R$. Let us call ω_1 the one that satisfies $\omega_1^2 > \frac{1}{2}$ and ω_2 the one such that $\omega_2^2 < \frac{1}{2}$. For the moment we do not specify the sign we take for each frequency. These signs will be determined below.

Now we want to obtain a real (and symplectic) change of variables such that the Hamiltonian (27) is reduced to its (real) normal form. The first step will be to look for the eigenvectors of the matrix M given by (29). To simplify the computation, we wil take advantage of the special form of this matrix. We denote by M_λ the matrix $M - \lambda I_4$, and we define the following splitting in 2×2 blocks:

$$M_\lambda = \begin{pmatrix} A_\lambda & I_2 \\ B & A_\lambda \end{pmatrix}, \qquad A_\lambda = \begin{pmatrix} -\lambda & 1 \\ -1 & -\lambda \end{pmatrix}, \qquad B = \begin{pmatrix} -\frac{1}{4} & a \\ a & \frac{5}{4} \end{pmatrix}.$$

Here, λ denotes one of the eigenvalues of the matrix M. The kernel of M_λ is now easy to find: to solve

$$\left(\begin{array}{cc} A_\lambda & I_2 \\ B & A_\lambda \end{array}\right)\left(\begin{array}{c} w_1 \\ w_2 \end{array}\right)=\left(\begin{array}{c} 0 \\ 0 \end{array}\right),$$

we can start by solving $(B-A^2)w_1=0$ and then $w_2=-Aw_1$ (note that the kernel of $(B-A^2)$ is trivial to find since it is a 2×2 matrix). In this way, we find the eigenvector $(2\lambda+a, \lambda^2-\frac{3}{4}, \lambda^2+a\lambda+\frac{3}{4}, \lambda^3+\frac{5}{4}\lambda+a)^\top$. Now, as the eigenvalues of M satisfy $\lambda=\sqrt{-1}\omega$, $\omega\in\mathbb{R}$, we obtain that the frequencies ω are determined by the equation

$$\omega^4-\omega^2+\frac{27}{16}-a^2=0. \tag{30}$$

We also apply $\lambda=\sqrt{-1}\omega$ to the expression of the eigenvector and, separating real and imaginary parts, we obtain that it can be expressed as $u+\sqrt{-1}v$, where

$$\left.\begin{array}{rcl} u(\omega) & = & \left(a, -\omega^2-\frac{3}{4}, -\omega^2+\frac{3}{4}, a\right)^\top \\ v(\omega) & = & \left(2\omega, 0, a\omega, -\omega^3+\frac{5}{4}\omega\right)^\top \end{array}\right\}. \tag{31}$$

We start considering the change of variables given by the matrix $C=(u_1,u_2,v_1,v_2)$, where u_j and v_j denote the values of u and v given by (31) corresponding to the frequencies ω_j, $j=1,2$. For the moment we do not specify which sign is taken for each frequency. In order to know whether C is simplectic or not, we check the property $C^\top JC=J$: a tedious but not difficult computation produces

$$C^\top JC=\left(\begin{array}{cc} 0 & D \\ -D & 0 \end{array}\right), \quad D=\left(\begin{array}{cc} d(\omega_1) & 0 \\ 0 & d(\omega_2) \end{array}\right).$$

where $d(\omega)=\omega(2\omega^4+\frac{1}{2}\omega^2-\frac{3}{4})$. Of course, to derive this expression you need to use the properties (30) and $\omega_1^2\omega_2^2=\frac{27}{16}-a^2$. Note that the zeros obtained in $C^\top JC$ and D were expected, due to the way we have constructed C. The only question was to know whether d were 1 or not. As it is not, we need to perform some scaling to the columns of C: let us define $s_j=\sqrt{d(\omega_j)}$, $j=1,2$ and let us redefine C as $(\frac{u_1}{s_1}, \frac{u_2}{s_2}, \frac{v_1}{s_1}, \frac{v_2}{s_2})$. This matrix is now symplectic, but we also want C to be real, that is, we want the values $d(\omega_j)$ to be positive. This will determine the signs we must choose for the frequencies ω_j. As $\omega_1^2<\frac{1}{2}$, if one wants $d(\omega_1)>0$ is necessary to take $\omega_1>0$ and, conversely, as $\omega_2^2<\frac{1}{2}$ implies that we must take $\omega_2<0$ in order to have $d(\omega_2)>0$. Hence, the change we have obtained is real, symplectic and it brings the Hamiltonian (27) into the real normal form

$$H_2=\frac{\omega_1}{2}(x^2+p_x^2)+\frac{\omega_2}{2}(y^2+p_y^2), \tag{32}$$

where we recall that $\omega_1>0$ and $\omega_2<0$.

In the paper we have used a complex normal form for H_2, because it allows to solve the homological equation that determines the generating function (see Section A.4) in a very easy way. Now it is not difficult to derive the change that brings (32) into complex normal form. We compose the complexifying change

$$\begin{array}{rclcrcl} x & = & \frac{q_1+\sqrt{-1}p_1}{\sqrt{2}}, & \quad & y & = & \frac{q_2+\sqrt{-1}p_2}{\sqrt{2}}, \\ p_x & = & \frac{\sqrt{-1}q_1+p_1}{\sqrt{2}}, & \quad & p_y & = & \frac{\sqrt{-1}q_2+p_2}{\sqrt{2}}, \end{array}$$

with the above-defined matrix C to produce the final change used in the paper:

$$\begin{pmatrix} \frac{a}{r_1}+\frac{2\omega_1}{r_1}\sqrt{-1} & \frac{2\omega_1}{r_1}+\frac{a}{r_1}\sqrt{-1} & \frac{a}{r_2}+\frac{2\omega_2}{r_2}\sqrt{-1} & \frac{2\omega_2}{r_2}+\frac{a}{r_2}\sqrt{-1} \\ \frac{-\omega_1^2-\frac{3}{4}}{r_1} & \frac{-\omega_1^2-\frac{3}{4}}{r_1}\sqrt{-1} & \frac{-\omega_2^2-\frac{3}{4}}{r_2} & \frac{-\omega_2^2-\frac{3}{4}}{r_2}\sqrt{-1} \\ \frac{-\omega_1^2+\frac{3}{4}}{r_1}+\frac{a\omega_1}{r_1}\sqrt{-1} & \frac{a\omega_1}{r_1}+\frac{-\omega_1^2+\frac{3}{4}}{r_1}\sqrt{-1} & \frac{-\omega_2^2+\frac{3}{4}}{r_2}+\frac{a\omega_2}{r_2}\sqrt{-1} & \frac{a\omega_2}{r_2}+\frac{-\omega_2^2+\frac{3}{4}}{r_2}\sqrt{-1} \\ \frac{a}{r_1}+\frac{-\omega_1^3+\frac{5}{4}\omega_1}{r_1}\sqrt{-1} & \frac{-\omega_1^3+\frac{5}{4}\omega_1}{r_1}+\frac{a}{r_1}\sqrt{-1} & \frac{a}{r_2}+\frac{-\omega_2^3+\frac{5}{4}\omega_2}{r_2}\sqrt{-1} & \frac{-\omega_2^3+\frac{5}{4}\omega_2}{r_2}+\frac{a}{r_2}\sqrt{-1} \end{pmatrix},$$

where

$$r_j = \sqrt{\omega_j\left(4\omega_j^4+\omega_j^2-\frac{3}{2}\right)}, \quad j=1,2.$$

Note that this matrix has been written assuming that the order of variables is for the initial variables (x, y, p_x, p_y) and (q_1, q_2, p_1, p_2) for the final ones. In the implementation of the software we have used the orders (x, p_x, y, p_y) and (q_1, p_1, q_2, p_2), that implies a permutation on this matrix.

B.2 The collinear points

Let us define, for $j = 1, 2$, γ_j as the distance from the smallest primary (the one of mass μ) to the point L_j, and γ_3 as the distance from the biggest primary to L_3. It is well-known (see, for instance, [52][12]) that γ_j is the only positive solution of the Euler quintic equation,

$$\begin{array}{c} \gamma_j^5 \mp (3-\mu)\gamma_j^4 + (3-2\mu)\gamma_j^3 - \mu\gamma_j^2 \pm 2\mu\gamma_j - \mu = 0, \quad j=1,2, \\ \gamma_j^5 + (2+\mu)\gamma_j^4 + (1+2\mu)\gamma_j^3 - (1-\mu)\gamma_j^2 - 2(1-\mu)\gamma_j - (1-\mu) = 0, \quad j=3, \end{array}$$

where the upper sign in the first equation is for L_1 and the lower one for L_2. These equations can be solved numerically by the Newton method, using the starting point $(\mu/3)^{1/3}$ for the first equation ($L_{1,2}$ cases), and $1-\frac{7}{12}\mu$ for the second one (L_3 case).

Next step would be to translate the origin to the selected point L_j, as it has been done for the triangular points. In this case, however, to have good numerical properties for the coefficients of the final expansions it is better to perform some scaling (see [39], [15], [23]). As the scalings are not symplectic transformations, let us consider the following process: first we write the differential equations related to (26) and then, on these equations, we

[12]Note that "our" L_1 and L_2 are swapped with respect to that reference. This lack of agreement for the definition of $L_{1,2}$ is rather common in the literature: usually, books on celestial mechanics use the same notation as [52] but books on astrodynamics use the one we have used.

perform the following substitution

$$
\begin{aligned}
X &= \mp\gamma_j x + \mu + \alpha_j, \\
Y &= \mp\gamma_j y, \\
Z &= \gamma_j z,
\end{aligned}
$$

where the upper sign corresponds to $L_{1,2}$, the lower one to L_3 and $\alpha_1 = -1+\gamma_1$, $\alpha_2 = -1-\gamma_2$ and $\alpha_3 = \gamma_3$. Note that the unit of distance is now the distance from the equilibrium point to the closest primary. Finally, it is not difficult to check that the differential equations obtained can be rewritten in Hamiltonian form, with Hamiltonian

$$H_{L_j} = \frac{1}{2}\left(p_x^2+p_y^2+p_z^2\right) + yp_x - xp_y - \sum_{n\geq 2} c_n(\mu)\rho^n P_n\left(\frac{x}{\rho}\right),$$

where $\rho^2 = x^2+y^2+z^2$, P_n is the Legendre polynomial of degree n and the coefficients $c_n(\mu)$ are given by

$$
\begin{aligned}
c_n(\mu) &= \frac{1}{\gamma_j^3}\left((\pm 1)^n\mu + (-1)^n\frac{(1-\mu)\gamma_j^{n+1}}{(1\mp\gamma_j)^{n+1}}\right), \quad j=1,2 \\
c_n(\mu) &= \frac{(-1)^n}{\gamma_j^3}\left(1-\mu+\frac{\mu\gamma_j^{n+1}}{(1+\gamma_j)^{n+1}}\right), \quad j=3.
\end{aligned}
$$

As usual, in the first equation, the upper sign is for L_1 and the lower one for L_2.

The linearization around the equilibrium point is given by the second order terms (linear terms must vanish) of the Hamiltonian that, after some rearranging, takes the form,

$$H_2 = \frac{1}{2}\left(p_x^2+p_y^2\right) + yp_x - xp_y - c_2x^2 + \frac{c_2}{2}y^2 + \frac{1}{2}p_z^2 + \frac{c_2}{2}z^2. \tag{33}$$

It is not difficult to derive intervals for the values of c_2 when $\mu \in [0, \frac{1}{2}]$ (see Figure 4). As $c_2 > 0$ (for the three collinear points), the vertical direction is an harmonic oscillator with frequency $\omega_2 = \sqrt{c_2}$. In what follows, we will focus on the planar directions, i.e.,

$$H_2 = \frac{1}{2}\left(p_x^2+p_y^2\right) + yp_x - xp_y - c_2x^2 + \frac{c_2}{2}y^2, \tag{34}$$

where, for simplicity, we keep the name H_2 for the Hamiltonian.

Now, we will proceed as in Section B.1. Let us define the matrix M as $J\mathrm{Hess}(H_2)$,

$$M = \begin{pmatrix} 0 & 1 & 1 & 0 \\ -1 & 0 & 0 & 1 \\ 2c_2 & 0 & 0 & 1 \\ 0 & -c_2 & -1 & 0 \end{pmatrix}. \tag{35}$$

The characteristic polynomial is $p(\lambda) = \lambda^4 + (2-c_2)\lambda^2 + (1+c_2-2c_2^2)$. Calling $\eta = \lambda^2$, we have that the roots of $p(\lambda) = 0$ are given by

$$\eta_1 = \frac{c_2 - 2 - \sqrt{9c_2^2 - 8c_2}}{2}, \qquad \eta_2 = \frac{c_2 - 2 + \sqrt{9c_2^2 - 8c_2}}{2}.$$

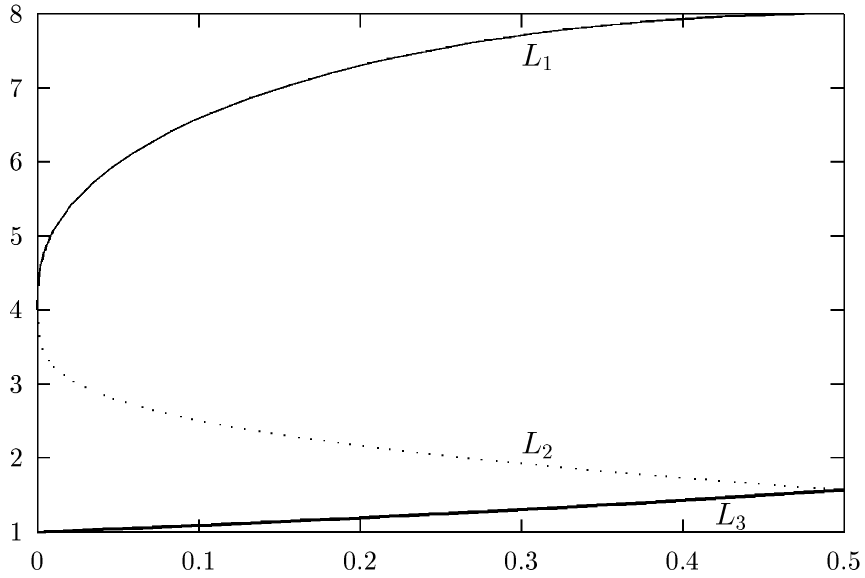

Figure 4: Values of $c_2(\mu)$, $\mu \in [0, \frac{1}{2}]$, for the cases $L_{1,2,3}$.

As $\mu > 0$, we have that $c_2 > 1$ that forces $\eta_1 < 0$ and $\eta_2 > 0$. This shows that the equilibrium point is a centre$\times$centre$\times$saddle. Thus, let us define ω_1 as $\sqrt{-\eta_1}$ and λ_1 as $\sqrt{\eta_2}$. For the moment, we do not specify the sign taken for each value (this will be discussed later on).

Now, as we did in the previous section, we want to find a symplectic linear change of variables casting (34) into its real normal form and, hence, we will look for the eigenvectors of matrix (35). As usual, we will take advantage of the special form of this matrix: if we denote by M_λ the matrix $M - \lambda I_4$, then

$$M_\lambda = \begin{pmatrix} A_\lambda & I_2 \\ B & A_\lambda \end{pmatrix}, \qquad A_\lambda = \begin{pmatrix} -\lambda & 1 \\ -1 & -\lambda \end{pmatrix}, \qquad B = \begin{pmatrix} 2c_2 & 0 \\ 0 & -c_2 \end{pmatrix}.$$

Now, the kernel of M_λ can be found using the same tricks as in the previous section: denoting as $(w_1^\top, w_2^\top)^\top$ the elements of the kernel, we start solving $(B - A^2)w_1 = 0$ and then $w_2 = -Aw_1$. Thus, the eigenvectors of M are given by $(2\lambda, \lambda^2 - 2c_2 - 1, \lambda^2 + 2c_2 + 1, \lambda^3 + (1 - 2c_2)\lambda)^\top$, where λ denotes the eigenvalues.

Let us start considering the eigenvectors related to ω_1. From $p(\lambda) = 0$, we obtain that ω_1 verifies

$$\omega_1^4 - (2 - c_2)\omega_1^2 + (1 + c_2 - 2c_2^2) = 0.$$

We also apply $\lambda = \sqrt{-1}\omega_1$ to the expression of the eigenvector and, separating real and imaginary parts as $u_{\omega_1} + \sqrt{-1}v_{\omega_1}$ we obtain

$$\begin{aligned} u_{\omega_1} &= (0, -\omega_1^2 - 2c_2 - 1, -\omega_1^2 + 2c_2 + 1, 0)^\top, \\ v_{\omega_1} &= (2\omega_1, 0, 0, -\omega_1^3 + (1 - 2c_2)\omega_1)^\top. \end{aligned}$$

Now, let us consider the eigenvalues related to $\pm\lambda_1$,

$$\begin{array}{rcl} u_{+\lambda_1} & = & (2\lambda, \lambda^2 - 2c_2 - 1, \lambda^2 + 2c_2 + 1, \lambda^3 + (1 - 2c_2)\lambda)^\top, \\ v_{-\lambda_1} & = & (-2\lambda, \lambda^2 - 2c_2 - 1, \lambda^2 + 2c_2 + 1, -\lambda^3 - (1 - 2c_2)\lambda)^\top. \end{array}$$

We consider, initially, the change of variables $C = (u_{+\lambda_1}, u_{\omega_1}, v_{-\lambda_1}, v_{\omega_1})$. To know whether this matrix is symplectic or not, we check $C^\top JC = J$. It is a tedious computation to see that

$$C^\top JC = \begin{pmatrix} 0 & D \\ -D & 0 \end{pmatrix}, \quad D = \begin{pmatrix} d_{\lambda_1} & 0 \\ 0 & d_{\omega_1} \end{pmatrix}.$$

This implies that we need to apply some scaling on the columns of C in order to have a symplectic change. The scaling is given by the factors

$$d_{\lambda_1} = 2\lambda_1((4 + 3c_2)\lambda_1^2 + 4 + 5c_2 - 6c_2^2), \quad d_{\omega_1} = \omega_1((4 + 3c_2)\omega_1^2 - 4 - 5c_2 + 6c_2^2).$$

Thus, we define $s_1 = \sqrt{d_{\lambda_1}}$ and $s_2 = \sqrt{d_{\omega_1}}$. As we want the change to be real, we have to ask $d_{\lambda_1} > 0$ and $d_{\omega_1} > 0$. It is not difficult to check that this condition is satisfied for $0 < \mu \leq \frac{1}{2}$ in all the points $L_{1,2,3}$, if $\lambda_1 > 0$ and $\omega_1 > 0$.

To obtain the final change, we have to take into account the vertical direction (z, p_z): to put it into real normal form we use the substitution

$$z \mapsto \frac{1}{\sqrt{\omega_2}} z, \qquad p_z \mapsto \sqrt{\omega_2} p_z.$$

This implies that the final change is given by the symplectic matrix

$$C = \begin{pmatrix} \frac{2\lambda}{s_1} & 0 & 0 & \frac{-2\lambda}{s_1} & \frac{2\omega_1}{s_2} & 0 \\ \frac{\lambda^2-2c_2-1}{s_1} & \frac{-\omega_1^2-2c_2-1}{s_2} & 0 & \frac{\lambda^2-2c_2-1}{s_1} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{\omega_2}} & 0 & 0 & 0 \\ \frac{\lambda^2+2c_2+1}{s_1} & \frac{-\omega_1^2+2c_2+1}{s_2} & 0 & \frac{\lambda^2+2c_2+1}{s_1} & 0 & 0 \\ \frac{\lambda^3+(1-2c_2)\lambda}{s_1} & 0 & 0 & \frac{-\lambda^3-(1-2c_2)\lambda}{s_1} & \frac{-\omega_1^3+(1-2c_2)\omega_1}{s_2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \sqrt{\omega_2} \end{pmatrix}$$

Finally, to produce the change that brings (33) into its complex normal form, we compose C with the same complexification as in the previous section.

To end this section, let us remark that here we have used the order (x, y, z, p_x, p_y, p_z) for the variables, while in the programs we have used the order (x, p_x, y, p_y, z, p_z). Of course, this implies a permutation on this matrix.

References

[1] Abraham R.H., Marsden J.E.: Foundations of Mechanics (2nd edition), Benjamin (1978).

[2] Arnol'd V.I.: Mathematical Methods of Classical Mechanics, Springer-Verlag, New York (1978).

[3] Arnol'd V. I.: *Instability of dynamical systems with several degrees of freedom*, Sov. Math. Dokl. 5:3 (1964), pp. 581–585.

[4] Arnol'd V.I., Avez A.: Ergodic Properties of Classical Mechanics, Benjamin, New York (1968).

[5] Arnol'd V.I., Kozlov V.V., Neishtadt A.I: Dynamical Systems III, Springer-Verlag (1988).

[6] Broucke R.: *A Fortran-based Poisson series processor and its applications in celestial mechanics*, Celestial Mechanics 45 (1989), pp. 255–265.

[7] Broucke R., Garthwaite K.: *A programming system for analytical series expansions on a computer*, Celestial Mechanics 1 (1969), pp. 271–284.

[8] Brumberg V.A., Tarasevich S.V., Vasiliev N.N.: *Specialized celestial mechanics systems for symbolic manipulation*, Celestial Mechanics 45 (1989), pp. 149–162.

[9] Bruno A.D.: Local Methods in Nonlinear Differential Equations, Springer-Verlag (1979).

[10] Celletti A., Chierchia L.: *Construction of analytic KAM surfaces and effective stability bounds*, Commun. Math. Phys. 118 (1988), pp 119–161.

[11] Celletti A., Giorgilli A.: *On the stability of the Lagrangian points in the spatial Restricted Three Body Problem*, Celestial Mechanics 50 (1991), pp. 31–58.

[12] Díez C., Jorba A., Simó C.: *A dynamical equivalent to the equilateral libration points of the Earth-Moon system*, Celestial Mechanics 50 (1991), pp. 13–29.

[13] Giorgilli A.: *A computer program for integrals of motion*, Comp. Phys. Comm. 16 (1979), pp. 331–343.

[14] Giorgilli A., Delshams A., Fontich E., Galgani L., Simó C.: *Effective stability for a Hamiltonian system near an elliptic equilibrium point, with an application to the Restricted Three Body Problem*, J. Differential Equations 77:1 (1989), pp. 167–198.

[15] Gómez G., Jorba A., Masdemont J., Simó C.: Study Refinement of Semi-Analytical Halo Orbit Theory, ESOC Contract 8625/89/D/MD(SC), Final Report (1991).

[16] Gómez G., Jorba A., Masdemont J., Simó C.: *A quasiperiodic solution as a substitute of L_4 in the Earth-Moon system*, in Proceedings of the 3rd International Symposium on Spacecraft Flight Dynamics, ESA Publications Division, ESTEC, Noordwijk, Holland (1991), pp. 35–41.

[17] Gómez G., Jorba A., Masdemont J., Simó C.: *A dynamical systems approach for the analysis of the SOHO mission*, in Proceedings of the 3rd International Symposium on Spacecraft Flight Dynamics, ESA Publications Division, ESTEC, Noordwijk, Holland (1991), pp. 449–454.

[18] Gómez G., Jorba A., Masdemont J., Simó C.: Study of Poincaré Maps for Orbits near Lagrangian Points, ESOC Contract 9711/91/D/IM(SC), Final Report (1993).

[19] Gómez G., Jorba A., Masdemont J., Simó C.: *Study of the transfer from the Earth to a halo orbit around the equilibrium point* L_1, Celestial Mechanics 56 (1993), pp. 541–562.

[20] Gómez G., Jorba A., Masdemont J., Simó C.: *Study of the transfer between halo orbits in the solar system*, Advances in the Astronautical Sciences 84 (1993), pp. 623–637.

[21] Gómez G., Llibre J., Martínez R., Simó C.: Station Keeping of Libration Point Orbits, ESOC Contract 5684/83/D/JS(SC), Final Report (1985).

[22] Gómez G., Llibre J., Martínez R., Simó C.: Study on Orbits near the Triangular Libration Points in the Perturbed Restricted Three-Body Problem, ESOC Contract 6139/84/D/JS(SC), Final Report (1987).

[23] Gómez G., Masdemont J., Simó C.: *Lissajous orbits around Halo orbits*, preprint (1997).

[24] Henrard J.: *A survey of Poisson series processors*, Celestial Mechanics 45 (1989), pp. 245–253.

[25] Jorba A., Llave R.: *Regularity properties of center manifolds and applications*, in preparation.

[26] Jorba A., Masdemont J.: *Nonlinear dynamics in an extended neighbourhood of the translunar equilibrium point*, in Hamiltonian Systems with Three or More Degrees of Freedom, Plenum Press (to appear).

[27] Jorba A., Masdemont J.: *Dynamics in the center manifold of the collinear points of the Restricted Three Body Problem*, preprint (1998).

[28] Jorba A., Simó C.: *Effective stability for periodically perturbed Hamiltonian systems*, in Hamiltonian Mechanics, Ed. J. Seimenis, Plenum Press, New York (1994), pp. 245–252.

[29] Jorba A., Villanueva J.: *Effective stability around periodic orbits of the spatial RTBP*, in Hamiltonian Systems with Three or more Degrees of Freedom, Plenum Press (to appear).

[30] Jorba A., Villanueva J.: *On the normal behaviour of partially elliptic lower dimensional tori of Hamiltonian systems*, Nonlinearity 10 (1997), pp. 783–822.

[31] Jorba A., Villanueva J.: *On the persistence of lower dimensional invariant tori under quasiperiodic perturbations*. Journal of Nonlinear Science 7 (1997) pp. 427–473.

[32] Jorba A., Villanueva J.: *Numerical computation of normal forms around some periodic orbits of the Restricted Three Body Problem*, to appear in Physica D.

[33] Kernighan B., Ritchie D: The C Programming Language (second edition), Prentice Hall, Englewood Cliffs, New Jersey (1988).

[34] Laskar J.: *Manipulation des series*, in D. Benest et C. Froeschlé (editors): Modern Methods in Celestial Mechanics, Editions Frontières (1990).

[35] Llave R., Marco, J. M., Moriyón, R.: *Canonical perturbation theory of Anosov systems and regularity results for the Livšic cohomology equation*, Ann. of Math. (2), 123(3) (1986) pp. 537–611.

[36] Llave R., Rana D.: *Accurate strategies for small denominator problems*, Bull. Amer. Math. Soc. 22 (1990), pp. 85–90.

[37] Marchal C.: *The quasi integrals*, Celestial Mechanics 21 (1980), pp. 183–191.

[38] Meyer K. R., Hall G. R.: Introduction to Hamiltonian Dynamical Systems and the N-Body Problem, Springer-Verlag (1992).

[39] Richardson, D. L.: *A note on a Lagrangian formulation for motion about the collinear points*, Celestial Mechanics 22 (1980), pp. 231–236.

[40] Ricklefs R., Jefferys W., Broucke R.: *A general precompiler for algebraic manipulation*, Celestial Mechanics 29 (1983), pp. 179–190.

[41] Schelter, W.: *SCC: An extension of Ansi C*. It can be retrieved by anonymous ftp from `ftp.math.utexas.edu:/pub`, file `scc.tar.gz`.

[42] Siegel C.L., Moser J.: Lectures on Celestial Mechanics, Springer-Verlag (1971).

[43] Simó C.: *Estabilitat de sistemes hamiltonians*, Mem. de la Real Acad. de Ciencias y Artes de Barcelona, Vol. XLVIII, no. 7 (1989).

[44] Simó C.: *Analytical and numerical computation of invariant manifolds*, in D. Benest et C. Froeschlé (editors): Modern Methods in Celestial Mechanics, Editions Frontières (1990), pp. 285-330.

[45] Simó C.: *Averaging under fast quasiperiodic forcing*, in J. Seimenis (editor): Hamiltonian Mechanics, Integrability and Chaotic Behaviour, vol. 331 of NATO Adv. Sci. Inst. Ser. B Phys. Plenum Press, New York (1994), pp. 13–34.

[46] Simó C.: *Effective computations in Hamiltonian dynamics*, in Cent ans après les Méthodes Nouvelles de H. Poincaré, Société Mathématique de France (1996), pp. 1–23.

[47] Simó C.: *Effective computations in celestial mechanics and astrodynamics*, lectures given at CISM Course on Modern Methods of Analytical Mechanics and their Applications, Udine, Italy (1997).

[48] Simó C.: *High precision numerical experiments of the splitting of separatrices for area preserving maps: Refined formulae and evidence of resurgence*, preprint (1998).

[49] Simó C.: *Methods of computation of invariant tori in Celestial Mechanics. Applications to the motion of small bodies*, to appear in A. Roy (editor): Proc. NATO ASI: The Dynamics of Small Bodies in the Solar System.

[50] Simó C., Gómez G., Jorba A., Masdemont J.: *The bicircular model near the triangular libration points of the RTBP*, in A.E. Roy and B.A. Steves (editors): From Newton to Chaos, Plenum Press, New York (1995), pp. 343–370.

[51] Stroustrup B: The C++ Programming Language (second edition), Addison Wesley, Reading, Massachusetts (1992).

[52] Szebehely V.: Theory of Orbits, Academic Press (1967).