Numerical computation of invariant KAM tori for the Sun-Jupiter-Saturn system using a variational principle

---

Ian JAUSLIN

Supervised by **Jacques LASKAR**

IMCCE, Observatoire de Paris
77 av. Denfert-Rochereau, 75014 Paris, France

Report of the internship for the curriculum of the CFP masters program, Quantum mechanics major, of ENS Paris

January-April 2012

Abstract

In this report, we describe a technique for computing quasi-periodic orbits of perturbed Hamiltonian systems numerically, and apply it to the Sun-Jupiter-Saturn system. Such a system has been studied by U. Locatelli and A. Giorgilli for reduced values of the planet's masses using a *standard* KAM approach. The method we use is based on a variational principle introduced by I.C. Percival, which has been proven to yield invariant KAM tori when applied to perturbed Hamiltonian systems by J. Moser, D. Salamon and E. Zehnder. The variational principle is formulated for a fixed frequency vector.

To find the extremum of the variational problem, we use a *quasi-Newton* algorithm, which has not converged for the realistic values of the masses of the planets. We discuss the reasons for this fact at the end of the report, and detail the next steps to be undertaken to solve the problems.

I wish to wholeheartedly thank Jacques Laskar, Mickael Gastineau, Nicolas Delsate, Alain Chenciner, Jacques Féjoz, Philippe Robutel, Alain Albouy and Laurent Niederman for very helpful discussions throughout the internship, as well as the entire team at the IMCCE institute for their precious help and support.

I am grateful to the administration officials at the ENS Paris and the Observatoire de Paris for their efforts organizing this project.

Table of contents:

Introduction

The dynamics of a single planet around a star is fairly simple, and has been understood since the works of J. Kepler in the XVI$^{\text{th}}$ century. A simple computation shows that the orbit of the planet is an ellipse whose parameters are easily determined from the initial conditions. However, the dynamics of two planets revolving around a star is extremely complicated, and has not yet been fully understood. If the planets did not interact, they would simply revolve around the star in two distinct ellipses, but when one takes their gravitational interaction into account, one finds that the motion can be fundamentally different from the non-interacting one. In fact, some trajectories of these three body problems are *chaotic*, in the sense that they behave so erratically that a very slight change in the initial conditions can radically alter their motion, which makes them very difficult to predict.

The question of which trajectories are *regular*, i.e. almost periodic, and which are *chaotic* is made all the more interesting by the fact that despite the efforts of some of the most renowned scientists of the XVIII$^{\text{th}}$ century, the stability of our Solar System has not been proven. While studying anomalies in the motion of Saturn and Jupiter, L. Euler, P.S. Laplace, J.L. Lagrange and S. Poisson have attempted (and succeeded to some extent) to understand many-planet motions using perturbation theories. U. le Verrier noticed in 1856 that the series on which the perturbation theories were based had large terms, and in the 1890's, H. Poincaré proved they were in fact divergent in most cases, which obliterated all hope for a simple proof of the stability of the Solar System. In 1954, A.N. Kolmogorov [Ko54] figured out which motions could be treated via a perturbation theory, and which could not. The proof he gave was somewhat incomplete, and was finished and perfected by V.I. Arnol'd [Ar63a] and J. Moser [Mo62] in the 1960's, giving birth to the so called *KAM theorem*. The answer to the question "Is the Solar System stable?" came in 1989, when J. Laskar showed, using a novel type of numerical integrations, that the system consisting of the Sun, Mercury, Venus, the Earth and Mars is chaotic. On the bright side, the effects can only begin to be seen on a time scale of 5 million years [La89], and even then, the effects would not give rise to cataclysmic events; however, J. Laskar and M. Gastineau subsequently showed [LG09] that a collision between Venus and the Earth, or even the ejection of Mars from the Solar System, could occur in the coming 5 billion years... This brief historical overview is based on [La10].

We restrict ourselves to the case of three bodies, to which we will refer as the *three body problem*. In the planetary case, where two of the bodies are planets, numerical computations show that the system is far more stable than in the Solar System's case. Its stability remains difficult to prove analytically though. The problem has been studied at length in Hamiltonian formulation using perturbation theories: e.g. in [Ar63b, Ro95 CC97, LG05] to name a few. In these references, the authors apply the constructive perturbative algorithm developed in the KAM theorem to find regular solutions, which turns out to be difficult, since the masses involved in planetary systems are too large to fit into the scope of the KAM theorem. For instance, in order to compute regular orbits for the Sun-Jupiter-Saturn system, U. Locatelli and A. Giorgilli [LG05] reduced the masses of both planets by a factor 1 000. The objective of this work is

to come at the problem using a different approach, and compute regular orbits for the Sun-Jupiter-Saturn system.

Instead of applying the KAM algorithm in Hamiltonian formulation, we search for the extremum of a variational problem, introduced by I.C Percival [Pe79], similar to the action principle of classical mechanics. We now give a quick overview of the method, the details of which are given in section 1. The regular motions of a Hamiltonian system are periodic or *quasi-periodic*, i.e. they can be expressed as a Fourier series with a finite number of frequencies (called the *frequency vector*), and evolve on a *torus* of phase space, i.e. are parametrized by angles defined modulo 2π. We fix the frequency vector a priori to some ω, and search for motions that evolve at the given frequency ω. The functional to be extremalized, to which we refer as the *action*, is defined on the space of tori, and it is such that its extremum is an *invariant torus*, in the sense that it is stable by the dynamics. The trajectories evolve on the extremal torus at the frequency ω. The action is defined by

$$S_\omega[\mathbf{q}] := \oint d\Phi \; L(\mathbf{q}(\Phi), D_\omega \mathbf{q}(\Phi))$$

where L is the Lagrangian of the system, $\mathbf{q}$ is a parametrization of an n-dimensional torus, and $D_\omega := \omega \cdot \nabla$. The existence of the extremum for close to integrable non-degenerate systems has been proven by J.N. Mather [Ma82a, Ma82b], J. Moser [Mo88], D. Salamon and E. Zehnder [SZ89].

In section 1 we recall a few basic notions of the theory of integrable systems, state and discuss the KAM theorem, and describe the variational principle in full detail. In section 2, we describe an extremalization algorithm applied to the Kepler problem, in order to introduce the three body problem, which is a perturbation of that system. We first briefly introduce the system and explain how to solve it analytically, then define a *Newton algorithm* to extremalize the action, discuss the difficulty of imposing an initial value for the trajectories, and discuss the results. In section 3, we present the three body problem in the Hill-Jacobi variables, set up the Newton algorithm and compute the derivatives of the action. The Newton algorithm requires too much time and memory, so we turn to two other algorithms described in section 4: the conjugate gradient algorithm, which has a simple structure and requires little memory, but is numerically unstable; and the quasi-Newton algorithm, which we then use to find the extremum of the action of the three body problem. It turns out that the algorithm fails to converge, so in section 5, we discuss a possible reason: the Kepler problem is degenerate, and thus the variational principle might have to be adapted for systems that are close to degenerate. We then present a simple far from degenerate system for which we have had positive results, and discuss the next steps to be taken.

The result of this work, which is a temporary result, is that Percival's variational principle can not be applied as such to the Sun-Jupiter-Saturn system. I suspect the problem is that the existence and uniqueness of the extremum have, to my knowledge, only been proven for non-degenerate systems [SZ89]. The problem arising from the degeneracy of the Kepler problem has been treated for the Hamiltonian formulation of

the KAM theorem [Ar63b, CC98], and may also be dealt with in Lagrangian formulation, perhaps by changing the variational principle.

1. Background

1.1. Integrable Hamiltonian systems

Consider a system with n degrees of freedom whose dynamics is given by a time-independent Hamiltonian $H(\mathbf{q};\mathbf{p})$, which is a function of n *positions* $\mathbf{q} = (q_1, \cdots, q_n)$ and n *momenta* $\mathbf{p} = (p_1, \cdots, p_n)$. The equations of motion are given by

$$\begin{cases} \dot{q}_i = \dfrac{\partial H}{\partial p_i}(\mathbf{q};\mathbf{p}) \\ \dot{p}_i = -\dfrac{\partial H}{\partial q_i}(\mathbf{q};\mathbf{p}) \end{cases} . \tag{1.1}$$

Liouville's theorem states that if such a system has n *independent* conserved quantities, then the equations of motion can be solved by quadratures, i.e. by calculating integrals and inverting functions. In that case the system is said to be *integrable*. Furthermore, the Arnol'd-Liouville theorem states [Ar88, Ar78, Ar63a] that if the system is bounded, then there exists a canonical change of variables to the so called *action-angle variables*

$$(\Phi;\mathbf{I}) = (\phi_1, \cdots, \phi_n; I_1, \cdots, I_n)$$

where the ϕ_i are variables on a circle i.e.

$$\phi_i + 2\pi \equiv \phi_i$$

and are such that H only depends on $\mathbf{I}$. Therefore

$$\begin{cases} \dot{\phi}_i = \dfrac{\partial H}{\partial I_i}(\mathbf{I}) \\ \dot{I}_i = -\dfrac{\partial H}{\partial \phi_i}(\mathbf{I}) = 0 \end{cases} . \tag{1.2}$$

so the I_i are conserved quantities, and thus $\dot{\phi}_i$ is constant, so

$$\phi_i(t) = \omega_i t + \phi_i^{(0)}$$

where

$$\omega_i := \frac{\partial H}{\partial I_i}(\mathbf{I}).$$

Thus the motion of an integrable system may be reduced to a collection of uniform motions on n circles at frequencies ω_i, or equivalently to a motion on a torus of dimension

n of frequency vector ω. If the quotients of all the ω_i are rational, then the trajectories on the torus are closed curves, so the motion is periodic. If the frequencies are *rationally independent*, i.e. if for $k \in \mathbb{Q}^n$,

$$(k \cdot \omega = 0) \Longrightarrow (k = 0)\,,$$

then the trajectories are dense in the torus, i.e. they come arbitrarily close to any point of the torus. In that case, the motion is called *quasi-periodic*.

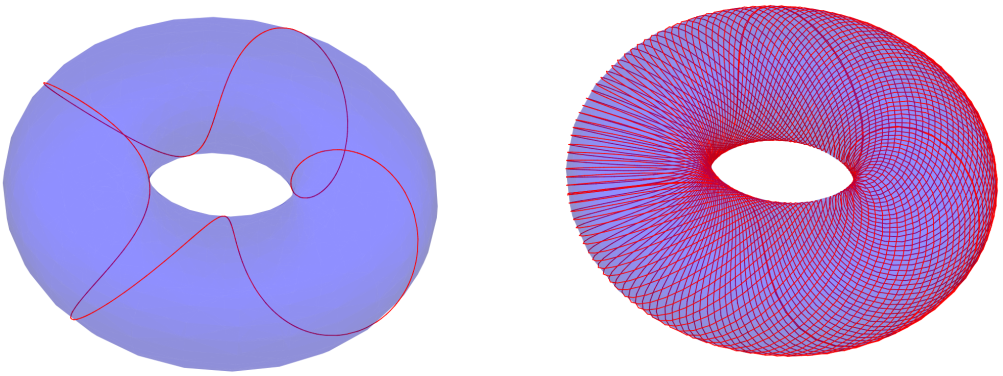


fig 1.1: Motions on a two-dimensional torus. Left: periodic orbit for $\frac{\omega_1}{\omega_2} = \frac{1}{4}$. Right: quasi-periodic orbit for $\frac{\omega_1}{\omega_2} = \frac{1}{\sqrt{10}}$, the trajectory covers the entire torus.

1.2. Perturbed Hamiltonian systems

We now consider a Hamiltonian of the form

$$H = H_0 + \epsilon H_1$$

where H_0 is integrable. As we have stated earlier, if $\epsilon = 0$, then phase space is covered by invariant tori, but if $\epsilon \neq 0$, then there will be trajectories that are neither periodic nor quasi-periodic. If ϵ is small, we expect that some of the trajectories will still be on tori. The question of finding which ones was answered by A.N. Kolmogorov, V.I. Arnol'd and J. Moser in the so-called *KAM theorem* [Ko54, Ar63a, Mo62], which states that there are invariant tori that are stable by a small perturbation, i.e. that there are frequencies ω such that all the trajectories that evolve at frequency ω in the non-interacting case remain quasi-periodic for small values of ϵ. These frequencies are those that verify the following *diophantine* condition:

Definition 1.1 : A vector $\omega \in \mathbb{R}^n$ is said to be *diophantine* if there exists $c > 0$ and $\eta > 0$ such that

$$\forall k \in \mathbb{Z}^n \setminus \{0\}, \quad |k \cdot \omega| \geqslant \frac{c}{\|k\|^\eta} \tag{1.3}$$

Essentially, diophantine frequency vectors are *far* from being *rationally dependent*. Rationally dependent frequencies are called *resonant*. The formulation of the theorem is as follows:

Theorem 1.1 [Ko54]: Consider a system with the Hamiltonian

$$H(\mathbf{q};\mathbf{p}) = H_0(\mathbf{q};\mathbf{p}) + \epsilon H_1(\mathbf{q};\mathbf{p})$$

such that H_0 is integrable. If H is *analytic* in $(\mathbf{q};\mathbf{p})$, and if H_0 is *non-degenerate* i.e.

$$\det\left(\frac{\partial^2 H_0}{\partial p_\alpha \partial p_\beta}(\mathbf{q};0)\right) \neq 0.$$

Then given a point in phase space $(\mathbf{q}_0;\mathbf{p}_0)$, and ω the frequency vector of the trajectory computed for $\epsilon = 0$, that starts at $(\mathbf{q}_0;\mathbf{p}_0)$; if ω is *diophantine* with parameters η and c, then there exists $\epsilon_0 > 0$, which may depend on η and c, such that for any $|\epsilon| < \epsilon_0$, there exists a trajectory starting from a point in the neighborhood of $(\mathbf{q}_0;\mathbf{p}_0)$ that is quasi-periodic, with a frequency vector equal to ω.

The idea of the proof of this theorem is given in appendix A2. Thus all the unperturbed trajectories on tori with diophantine frequency vectors will remain regular in the perturbed case. The tori of diophantine frequencies are called *stable*. The other tori will be *destroyed* by the perturbation, which leads to chaotic trajectories.

Notice that the constants c and η in (1.3) depend on ϵ, and we expect that as ϵ gets larger, c increases and η decreases, making (1.3) more and more constraining, thus breaking more and more tori. However, *most* of the tori are stable if η is large enough. More precisely, one can prove that if $\eta > n-1$, for any $c > 0$, the set

$$\left\{\omega \in \mathbb{R}^n \text{ such that } \exists k \in \mathbb{Z}^n \setminus \{0\},\ |k\cdot\omega| < \frac{c}{\|k\|^\eta}\right\}$$

of frequencies that do not verify (1.3) has measure 0 (see appendix A1), hence if ϵ is small enough, then almost all the tori are preserved.

The goal of this work is to find invariant tori for the Sun-Jupiter-Saturn system, where the interaction between Jupiter and Saturn is considered as a perturbation of the integrable dynamics of the planets around the Sun. We therefore need an explicit algorithm to compute KAM tori. One way of doing this comes from Kolmogorov's proof of the KAM theorem [Ko54] (see appendix A2), which gives an explicit construction of the invariant tori. The idea is to find a canonical change of variables $(\mathbf{q};\mathbf{p}) \mapsto (\mathbf{Q};\mathbf{P})$ to transform a Hamiltonian of the form

$$H(\mathbf{q};\mathbf{p}) = (m + \omega\cdot(\mathbf{p}-\mathbf{I})) + \epsilon\,(A(\mathbf{q}) + \mathbf{B}(\mathbf{q})\cdot(\mathbf{p}-\mathbf{I})) + O(|\mathbf{p}-\mathbf{I}|^2) \tag{1.4}$$

into the so called *Kolmogorov normal form*

$$H(\mathbf{Q};\mathbf{P}) = M(\epsilon) + \omega \cdot (\mathbf{P} - \mathbf{I}) + O(|\mathbf{P} - \mathbf{I}|^2) \tag{1.5}$$

thus the trajectories passing through $\mathbf{P} = \mathbf{I}$ would be solutions of the equations

$$\begin{cases} \dot{\mathbf{Q}} = \omega \\ \dot{\mathbf{P}} = 0 \end{cases}$$

and thus would be restricted to an invariant torus of frequency ω. The method used to find such a canonical change of variables is a *Newton method* which consists in a sequence of canonical transformations that converges super-linearly to the one that transforms (1.4) into (1.5).

Kolmogorov's method was applied to the Sun-Jupiter-Saturn system by U. Locatelli & A. Giorgilli [LG05], albeit with smaller masses than those measured for Jupiter and Saturn. In this work, we search for invariant tori using a different approach, based on a variational principle analogous to the action principle of classical mechanics.

1.3. A variational principle on tori

The action principle states that physical trajectories are those that extremalize the action functional. We have seen in the previous sections that the natural objects appearing in nearly integrable Hamiltonian systems are tori, and not trajectories. We shall re-formulate the action principle into a variational principle on an action which is a function of tori. This principle is based on works by Percival [Pe79, Ar88].

Since Tori are continuous deformations of the unit torus $\mathbb{T}^n$ (i.e. of the cartesian product of n unit circles), we can parametrize each of them by two continuous functions, $\mathbf{q}(\Phi)$ and $\mathbf{p}(\Phi)$ defined for Φ in $\mathbb{T}^n$. In this work we will suppose that $\mathbf{q}$ and $\mathbf{p}$ are analytic functions. We wish to find a condition on $\mathbf{q}$ and $\mathbf{p}$ that ensures that the torus they parametrize is invariant. First, we define precisely what we mean by *invariant*:

> **Definition 1.2** : A torus parametrized by $(\mathbf{q};\mathbf{p})$ is said to be *invariant* if $\exists \omega \in \mathbb{R}^n$ such that $\forall \Phi_0 \in \mathbb{T}^n$,
>
> $$(\mathbf{q}(\Phi_0 + \omega t); \mathbf{p}(\Phi_o + \omega t))$$
>
> is a solution of Hamiton's equations. In that case ω is called the *frequency vector* associated to the torus.

We prove the following lemma:

> **Lemma 1.1** [Pe79]: Let L be the Lagrangian of the system. A torus parametrized by two continuous functions $\mathbf{q}$ and $\mathbf{p}$ is

invariant by the dynamics if and only if

$$\begin{cases} D_\omega \dfrac{\partial L}{\partial \dot{q}_i}\left(\mathbf{q}(\Phi), D_\omega \mathbf{q}(\Phi)\right) = \dfrac{\partial L}{\partial q_i}\left(\mathbf{q}(\Phi), D_\omega \mathbf{q}(\Phi)\right) \\ \mathbf{p}(\Phi) = \dfrac{\partial L}{\partial \dot{\mathbf{q}}}\left(\mathbf{q}(\Phi), D_\omega \mathbf{q}(\Phi)\right) \end{cases} \tag{1.6}$$

where

$$D_\omega := \omega \cdot \nabla_{\mathbf{q}}. \tag{1.7}$$

The proof is straightforward, keeping in mind that Hamilton's equations (1.2) are equivalent to the Euler-Lagrange equation

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} = \frac{\partial L}{\partial q_i}. \tag{1.8}$$

We define the action S_ω as a functional of $\mathbf{q}(\Phi)$ by

$$S_\omega[\mathbf{q}] := \oint d\Phi \; L(\mathbf{q}(\Phi), D_\omega \mathbf{q}(\Phi)) = \frac{1}{(2\pi)^n} \int_{-\pi}^{\pi} d\phi_1 \cdots \int_{-\pi}^{\pi} d\phi_n \; L(\mathbf{q}(\Phi), D_\omega \mathbf{q}(\Phi)). \tag{1.9}$$

A simple computation proves the following theorem:

> **Theorem 1.2** [Pe79]: A torus parametrized by $(\mathbf{q}; \mathbf{p})$ is invariant if and only if $\mathbf{q}$ extremalizes the action S_ω, i.e. if for any analytic function $\delta\mathbf{q}$ from $\mathbb{T}^n$ to $\mathbb{T}^n$,
>
> $$\frac{d}{d\epsilon} S_\omega[\mathbf{q} + \epsilon\delta\mathbf{q}] = 0$$
>
> and
>
> $$\mathbf{p}(\Phi) = \frac{\partial L}{\partial \dot{\mathbf{q}}}\left(\mathbf{q}(\Phi), D_\omega \mathbf{q}(\Phi)\right).$$

This theorem provides a variational principle for invariant tori, that is analogous to the action principle of classical mechanics. There is however a fundamental difference: whereas the classical action is extremalized with fixed initial and final positions, S_ω is extremalized with a fixed frequency vector ω, but none of the points of the torus are a priori imposed.

There are a few technical details one then has to take into account. Such a procedure can only yield a unique result if the system is such that all the invariant tori in phase space have different frequencies. Conversely, an extremum of S_ω can only exist if there exists an invariant torus of frequency ω. If one wants to use this formalism when this is not the case, e.g. for a two-dimensional harmonic oscillator, one must chose a point on the torus and extremalize S_ω with that point fixed. In that case, one must make sure that there exists a torus in phase space of frequency ω passing through the imposed point, which may be a difficult problem.

For example, consider the simple case of a planet revolving around a star, to which will shall refer as a *Kepler problem*. This system is reducible to one degree of freedom, and is thus integrable. The trajectories are contained within a plane and are ellipses, that depend on two parameters: the semi-major axis a and the eccentricity e. There is a simple relation between the frequency of a trajectory and the semi-major axis, therefore two tori that have different eccentricities but the same semi-major axis will have the same frequency. To find invariant tori of the Kepler problem, a point of the torus we are searching for must be fixed.

The existence and uniqueness of the extremum of (1.9) has been studied by J.N. Mather and J. Moser [Ma82a, Ma82b, Mo86]. Mather proved that for Hamiltonian systems with two degrees of freedom, the extremum of the action exists [Ma82b] and is unique [Ma82a]. Moser extended this result to a more general form of variational problems [Mo86]. In these works, the authors proved that the existence and unicity of the solution of the variational problem is not conditioned by the diophantine condition (1.3), therefore the action has an extremum even if there is no invariant torus. However, in such cases, the extremum of the action is not continuous, so it is not a torus, but merely an invariant subset of phase space. In fact, such an extremum is a transversal Cantor set, i.e. it has an infinite, non-countable number of discontinuity points. The set thus obtained is called an *Aubry-Mather set*, or a *Cantorus*.

Thus there is a close analogy between the extremalization of the action on trajectories and on tori. But instead of having regular and chaotic trajectories, this formalism yields invariant tori and invariant Cantori.

Moreover, J. Moser proved an analog to the KAM theorem for the extremalization of the action on tori [Mo86, Mo88], which was generalized to Hamiltonian systems in arbitrary dimensions by D. Salamon and E. Zehnder [SZ89]. The statement of this theorem is:

> **Theorem 1.3** [SZ89]: Given an $\omega \in \mathbb{R}^n$. Consider the Lagrangian
>
> $$L(\mathbf{q};\dot{\mathbf{q}}) = L_0(\mathbf{q};\dot{\mathbf{q}}) + \epsilon L_1(\mathbf{q};\dot{\mathbf{q}})$$
>
> and an analytic function $\mathbf{q}_0$ from $\mathbb{T}^n$ to $\mathbb{T}^n$ that is an extremum of the action
>
> $$S_\omega^{(0)}[\mathbf{q}] := \oint d\Phi \; L_0(\mathbf{q}(\Phi), D_\omega \mathbf{q}(\Phi)).$$
>
> If L is *analytic* in $(\mathbf{q};\dot{\mathbf{q}})$, $(L_0, \mathbf{q}_0)$ is *non-degenerate* (see the definition below), and ω is *diophantine*, then there exists $\epsilon_0 > 0$ such that if $|\epsilon| < \epsilon_0$, there exists a locally unique analytic $\mathbf{q}$ from $\mathbb{T}^n$ to $\mathbb{T}^n$ that extremalizes the action
>
> $$S_\omega[\mathbf{q}] := \oint d\Phi \; L(\mathbf{q}(\Phi), D_\omega \mathbf{q}(\Phi)).$$

Definition 1.3 : $(L_0, \mathbf{q}_0)$ is said to be *non-degenerate* if defining $\mathfrak{q}_0(\Phi)$ as the Jacobian matrix of $\mathbf{q}_0(\Phi)$, $\mathfrak{q}_0(\Phi)^T$ as its transpose, and $a(\Phi)$ as the matrix

$$a(\Phi) := \mathfrak{q}_0(\Phi)^T \frac{\partial^2 L_0}{\partial \dot{\mathbf{q}} \partial \dot{\mathbf{q}}}(\mathbf{q}_0(\Phi); D_\omega \mathbf{q}_0(\Phi))\mathfrak{q}_0(\Phi)$$

we have

$$\begin{cases} \det\left(a(\Phi)\right) \neq 0 \\ \\ \det\left(\oint d\Phi \; a(\Phi)^{-1} \neq 0\right) \end{cases}$$

Essentially the theorem entails that if the L_0 is the Lagrangian of an integrable system, we can change variables to the action-angle variables, in which the unperturbed motions will be trivial, and thus

$$\mathbf{q}_0(\Phi) = \Phi$$

will extremalize the unperturbed action. The theorem then proves the existence of an analytic extremum $\mathbf{q}$ of (1.9), provided that the perturbation is small enough, that the unperturbed Lagrangian is non-degenerate, and that ω is diophantine.

2. Preliminary: numerical solution of the Kepler problem

The goal of this work is to compute invariant tori numerically by extremalizing an action functional. This is fundamentally different from integrating a differential equation numerically using for example the Runge-Kutta method, since the goal is to find a function extremalizing the action all at once, rather than constructing a solution of a differential equation step by step. To check whether it is viable to search numerically for extrema of an action functional, we shall first apply such a method to the Kepler problem, which is integrable.

2.1. The Kepler problem

In this section, we define the Kepler problem and show that it is integrable and how to compute its solution.

We consider the problem of a planet revolving around a star, which can be seen as a point mass in a potential proportional to r^{-1}. The motion is planar, and in polar coordinates, the Hamiltonian can be written as

$$H(r, \theta; p, g) := \frac{p^2}{2\beta} + \frac{g^2}{2\beta r^2} - \frac{\mu\beta}{r} \tag{2.1}$$

where β is the mass of the planet, μ is the mass of the star multiplied by the gravitational constant $\mathcal{G}$, p is the momentum

$$p = \beta \dot{r}$$

and g is the angular momentum

$$g = \beta r^2 \dot{\theta}.$$

Since H does not depend on θ, g is a constant, thus the system can be reduced to one degree of freedom. The Lagrangian of the system is the Legendre transform of (2.1):

$$L(r; \dot{r}) = \frac{1}{2}\beta \dot{r}^2 - \frac{g^2}{2\beta r^2} + \frac{\mu\beta}{r} \tag{2.2}$$

and the action functional is

$$S[r] := \int dt \; L(r(t); \dot{r}(t)). \tag{2.3}$$

The system is integrable, in fact, one can easily prove, e.g. using the Runge-Lenz invariant vector, that the trajectories are ellipses given by the equation

$$r = \frac{a(1-e^2)}{1 + e\cos\theta}$$

where a is the *semi-major axis* and e the *eccentricity*. The angular momentum can be computed as a function of a and e:

$$g = \beta\sqrt{\mu a(1-e^2)}.$$

Furthermore, it is a well known fact that the *mean anomaly* M, i.e. the area covered by the position vector between the times 0 and t verifies

$$\dot{M}(t) = \sqrt{\frac{\mu}{a^3}} = cst.$$

In fact, one can show that there exists a canonical change of variables from $(r, \theta; p, g)$ to the *Delaunay variables* $(M, \varpi; L := \beta\sqrt{\mu a}, g)$ where ϖ is the *argument of the perihelion*, i.e. the angle θ of the closest point of the trajectory to the star; in terms of which the Hamiltonian is simply

$$H = -\frac{\mu^2\beta}{2L^2} \tag{2.4}$$

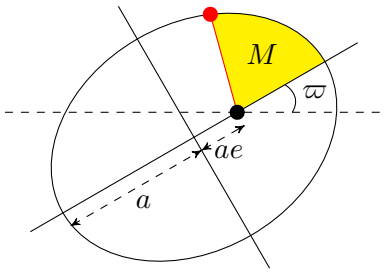


fig 2.1: Definition of the *semi major axis* a, the *eccentricity* e, the *mean anomaly* M and the *argument of the perihelion* ϖ.

Thus the Delaunay variables are action-angle variables for the Kepler problem. The Hamiltonian is *degenerate* and there is only one frequency

$$\omega = \sqrt{\frac{\mu}{a^3}}. \tag{2.5}$$

The change of variables from the polar to the Delaunay variables is implicit and difficult to manipulate, so in order to find the analytical solution of the Kepler problem, we introduce a new angle, the *eccentric anomaly* E, defined by

$$r =: a(1 - e\cos E)$$

which verifies the so-called *Kepler equation*

$$E = M + e\sin E. \tag{2.6}$$

Finding the explicit time dependence of r is thus reduced to solving (2.6). One can prove that the solution of the Kepler equation is given by the limit of the sequence of functions (E_n) defined by

$$E_{n+1}(M) = M + e\sin(E_n(M))$$

which converges uniformly. Thus

$$r(M) = a\,(1 - e\cos(E(M))) \tag{2.7}$$

which gives $r(t)$ using $\dot{M} = \omega$.

2.2. The Newton algorithm

We now describe how to compute the extremum of the action functional. We express r as a Fourier series

$$r(M) = \sum_{k=-\infty}^{\infty} r_k e^{ikM}$$

thus the action S can be seen as a function of the r_k. Its extremum is given by the set of r_k's such that

$$\partial_k S := \frac{\partial S}{\partial r_k} = 0. \tag{2.8}$$

In order to model this data on a computer, we only consider the k's such that $|k| \leqslant k_m$. To find the family of r_k's satisfying (2.8), we use a *Newton algorithm*, which consists in starting with a good approximation of the extremum, and improving it iteratively. We start with a trial function $r^{(0)}$, and compute the n-th improvement $r^{(n)}$ by imposing

$$\partial_k S\left(r^{(n+1)}\right) = O\left(\left\|r^{(n+1)} - r^{(n)}\right\|^2\right). \tag{2.9}$$

If S is sufficiently regular, we can express $\partial_k S\left(r^{(n+1)}\right)$ using a Taylor series, and we find

$$r^{(n+1)} = r^{(n)} - \left(D^2 S\left(r^{(n)}\right)\right)^{-1} \cdot \partial_k S\left(r^{(n)}\right) \tag{2.10}$$

where $D^2 S$ is the *Hessian* of the action, i.e. the matrix of its second derivatives. Because of condition (2.9), the Newton algorithm is *quadratically convergent*.

To implement the Newton algorithm, we must compute the gradient and the Hessian of the action. We use the fact that

$$\partial_k r = e^{ik\omega t},\ \partial_k \dot{r} = ik\omega e^{ikM}$$

and we introduce the notation

$$\langle f \rangle_{-k} = \int_0^{2\pi} dM\ e^{ikM} f(M)$$

so we can rewrite

$$S = \int_0^{2\pi} dM\ L(r(M), \dot{r}(M)) = \langle L \rangle_0 .$$

Therefore,

$$\partial_k S = \langle \partial_k L \rangle_0 = \omega^2 k^2 \beta r_{-k} + \left\langle \frac{g^2 - \beta^2 \mu r}{\beta r^3} \right\rangle_{-k} \tag{2.11}$$

and

$$\partial_k \partial_l S = \langle \partial_k \partial_l L \rangle_0 = \omega^2 k^2 \beta \delta_{k,-l} - \left\langle \frac{3g^2 - 2\beta^2 \mu r}{\beta r^4} \right\rangle_{-k-l}. \tag{2.12}$$

The Newton algorithm is simple, quadratically convergent, but it requires the inversion of a matrix. It is a generic algorithm to find a zero of a function and does not use the fact that the function we are studying is a gradient. There are other algorithms, that make use of this property, and are faster and more efficient. However, many require the extremum to be either a maximum or a minimum. In the problem that we are considering, we attempt to find the solution of the differential equation (1.6), that

happens to be the *extremum* of the action (1.9), which is not necessarily a maximum or a minimum: it may be a saddle point. In fact, using (2.12), we may see that we are here in the latter case: we approximate (2.12) by neglecting all the terms involving $\langle\cdot\rangle_k$ with $k \neq 0$. This is a reasonable approximation since analytic functions have Fourier coefficients that decay exponentially. The approximated Hessian is diagonal, so its sign is determined by the sign of its elements. We use the following estimates:

$$r \approx a \text{ and } g^2 = \beta^2 \mu a (1 - e^2)$$

thus

$$\partial_k \partial_{-k} S \approx \beta \frac{\mu}{a^3} \left(k^2 - (1 - 3e^2)\right)$$

so $\partial_0 \partial_0 S < 0$, but $\partial_2 \partial_2 S > 0$. Thus the Hessian of the action is neither positive nor negative definite, so the extremum of S is neither a maximum nor a minimum, but a saddle point.

2.3. Initial values

Since we are currently searching for trajectories of the Kepler problem, it is necessary to specify initial values for the solution we are looking for. This would remain true if we were searching for tori of fixed frequency, since the frequency is degenerate. This can easily be seen by the fact that there are two degrees of freedom for picking a torus: the semi-major axis and the eccentricity, whereas fixing the frequency alone can only fix one of these quantities. In fact, using the simple relation

$$\omega = \sqrt{\frac{\mu}{a^3}}$$

the frequency only sets the semi-major axis, and the eccentricity remains free. Thus, if we were looking for invariant tori of a fixed frequency, we would need to pin down a point of the torus to find a unique solution.

To simplify, we look for trajectories that start at $t = 0$ at the *perihelion*, i.e. the point of the trajectory closest to the star, Thus we impose $\dot{r}(0) = 0$, which we achieve by imposing $r_k = r_{-k}$. To fix $r(0)$, we fix a and e, and using the definition of e from fig. 2.1, we have

$$r(0) = a(1 - e). \tag{2.13}$$

Imposing (2.13) is more subtle than it seems. The simplest way of constraining the solution is to impose $r(0)$ by imposing the value of r_0. Explicitly, we consider S as a function of the r_k such that $k > 0$, and define

$$r(M) := a(1 - e) + \sum_{k=1}^{k_m} r_k \left(e^{ikM} + e^{-ikM} - 2\right) \tag{2.14}$$

thus (2.11) and (2.12) would become

$$\partial_k S = 2\omega^2 k^2 \beta r_k + 2\left\langle \frac{g^2 - \beta^2 \mu r}{\beta r^3} \right\rangle_k - 2\left\langle \frac{g^2 - \beta^2 \mu r}{\beta r^3} \right\rangle_0 \tag{2.15}$$

and

$$\partial_k \partial_l S = 2\omega^2 k^2 \beta \delta_{k,l} - 2\langle V_0 \rangle_{k+l} - 2\langle V_0 \rangle_{k-l} + 4\langle V_0 \rangle_k + 4\langle V_0 \rangle_l - 4\langle V_0 \rangle_0 \tag{2.16}$$

where

$$V_0 := \frac{3g^2 - 2\beta^2 \mu r}{\beta r^4}.$$

However, such an algorithm can (and for some values of e, does) converge to an r that extremalizes the action, but is not a solution of the Euler-Lagrange equations. We know however, that this cannot occur as long as the Lagrangian is $\mathcal{C}^1$, so it must be an artifact of the numerical integration. Indeed, this situation is created by the truncation of k. The fact that r extremalizes the action means that $\partial_k S(r) = 0$, which according to (2.15) is equivalent to

$$2\omega^2 k^2 \beta r_k + 2\left\langle \frac{g^2 - \beta^2 \mu r}{\beta r^3} \right\rangle_k = 2\left\langle \frac{g^2 - \beta^2 \mu r}{\beta r^3} \right\rangle_0$$

or in other words $\forall k \in \mathbb{Z}$,

$$\left\langle \frac{d}{dt}\frac{\partial L}{\partial \dot{r}} - \frac{\partial L}{\partial r} \right\rangle_k = \left\langle \frac{d}{dt}\frac{\partial L}{\partial \dot{r}} - \frac{\partial L}{\partial r} \right\rangle_0. \tag{2.17}$$

The only way (2.17) can be verified is if

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{r}} - \frac{\partial L}{\partial r} = 0$$

or if it is a Dirac distribution. If L is $\mathcal{C}^1$, the latter case cannot occur, so the Euler-Lagrange equation must be verified, but since we neglected all the k larger than k_m, the algorithm can produce r's verifying (2.17) but not the Euler-Lagrange equation.

We must find another way of imposing the initial condition. Fortunately, the extrema of the action have a remarkable property: an extremum on the sub-variety $\mathcal{V}$ of r's that verify the initial condition is an extremum on the entire variety of r's. This is a trivial consequence of the fact that an extremum on $\mathcal{V}$ is a solution of the Euler-Lagrange equation, regardless of the initial condition. Thus instead of searching for extrema of the action among the r's verifying the initial condition, we may look for extrema of the action that also obey the initial condition. We may do this by introducing a Lagrange multiplier λ, and extremalize

$$\Lambda(r, \lambda) := S(r) + \frac{\lambda}{2}\left(r_0 + 2\sum_{k=1}^{k_m} r_k - a(1-e) \right)^2 + \frac{\lambda^3}{6}. \tag{2.18}$$

Indeed, the gradient of Λ is given by

$$\begin{cases} \partial_k \Lambda = \partial_k S + 2\lambda \left(r_0 + 2 \sum_{k=1}^{k_m} r_k - a(1-e) \right) \\ \partial_\lambda \Lambda = \frac{1}{2} \left(r_0 + 2 \sum_{k=1}^{k_m} r_k - a(1-e) \right)^2 + \frac{\lambda^2}{2} \end{cases}$$

Thus if Λ is extremalized by (r, λ), then $\lambda = 0$, r verifies the initial condition, and $\partial_k S(r) = 0$.

In practice, extremalizing (2.18) is substantially longer than using (2.14), so we implemented an algorithm in which we first attempt to find the solution using (2.14) and switch to (2.18) only if we converge to a wrong solution (see program P1).

2.4. Discussion of the results

We implemented the Newton algorithm for the Kepler problem using a high-level computing language called TRIP, which was developed by M. Gastineau and J. Laskar [GL12, GL10]. TRIP was built to manipulate series efficiently and easily, and is thus a shoe-in for dealing with the Fourier series that arise in our problem. The code is given in program P1.

We pick a unit system in which $a = \beta = \omega = \mu = 1$. We ran the algorithm picking various values for the eccentricity e. The cutoff k_m is chosen so that the k_m-th Fourier component of the analytical solution of the Kepler problem is smaller than a given numerical error, e.g. 10^{-20}. This gives us control over the error made by the algorithm.

The algorithm converges in under 20 iterations for $e < 0.55$. After that, we tried a hundred different values between $e = 0.55$ and $e = 0.9$, for which the algorithm converged, sometimes after a few hundred iterations.

All in all, using a variational principle to find solutions of the Kepler problem works very well, especially for small eccentricities. We shall now use a similar algorithm for the non-integrable planar three body problem.

3. The three body problem

3.1. Hill-Jacobi variables

We now derive the Hamiltonian of the planar three body problem in the *Hill-Jacobi* variables, and prove that the system is reducible to three degrees of freedom.

Let m_0, m_1 and m_2 be the masses of the Sun, Jupiter and Saturn respectively. In cartesian variables $(\vec{u}_0,\ \vec{u}_1, \vec{u}_2; \tilde{u}_0, \tilde{u}_1, \tilde{u}_2)$ where

$$\tilde{u}_i = \frac{\dot{\vec{u}}_i}{m_i}$$

the Hamiltonian is

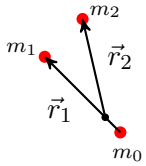
$$H(\vec{u}; \tilde{u}) := \sum_{i=1}^{3} \frac{\tilde{u}_i^2}{2m_i} - \sum_{i<j} \frac{\mathcal{G} m_i m_j}{\|\vec{u}_i - \vec{u}_j\|}. \tag{3.1}$$

We change variables canonically to the Jacobi variables defined by

$$\begin{cases} \vec{r}_0 := \vec{u}_0 \\ \vec{r}_1 := \vec{u}_1 - \vec{u}_0 \\ \vec{r}_2 := \vec{u}_2 - \delta_1 \vec{u}_1 - \delta_0 \vec{u}_0 \end{cases} \quad \text{and} \quad \begin{cases} \tilde{r}_0 = \tilde{u}_2 + \tilde{u}_1 + \tilde{u}_0 \\ \tilde{r}_1 = \delta \tilde{u}_2 + \tilde{u}_1 \\ \tilde{r}_2 = \tilde{u}_2 \end{cases} \tag{3.2}$$

where

$$\delta_1 := \frac{m_1}{m_0 + m_1} \text{ and } \delta_0 := \frac{m_0}{m_0 + m_1}$$



In these variables, $\tilde{r}_0$ is the momentum of the center of mass, which we can set to 0. Thus (3.1) becomes

$$H = K_1 + K_2 + H_{int} \tag{3.3}$$

where

$$\begin{cases} K_i := \dfrac{\tilde{r}_i^2}{2\beta_i} - \dfrac{\mu_i \beta_i}{\|\vec{r}_i\|} \\ H_{int} := \dfrac{\mu_2 \beta_2}{\|\vec{r}_2\|} - \dfrac{\mu_2 \beta_2}{\|\vec{r}_2 + \delta \vec{r}_1\|} - \dfrac{\mu}{\|\vec{r}_2 - (1-\delta)\vec{r}_1\|} \end{cases}$$

and

$$\begin{cases} \beta_1 := \dfrac{m_0 m_1}{m_0 + m_1} \quad \beta_2 := \dfrac{(m_0 + m_1) m_2}{m_0 + m_1 + m_2} \\ \mu_1 := \mathcal{G}(m_0 + m_1) \quad \mu_2 := \mathcal{G}(m_0 + m_1 + m_2)(1 - \delta) \\ \mu := \mathcal{G} m_1 m_2 \end{cases} \tag{3.4}$$

Notice that H does not depend on $\vec{r}_0$, so we may reduce our system to the variables $(\vec{r}_1, \vec{r}_2; \tilde{r}_1, \tilde{r}_2)$. We then define the Hill-Jacobi variables $(r_1, r_2, v_1, v_2; p_1, p_2, g_1, g_2)$ by

$$\vec{r}_i = r_i \begin{pmatrix} \cos(v_i) \\ \sin(v_i) \end{pmatrix} \text{ and } \begin{pmatrix} p_i \\ g_i \end{pmatrix} = \begin{pmatrix} \cos(v_i) & \sin(v_i) \\ -r_i \sin(v_i) & r_i \cos(v_i) \end{pmatrix} \tilde{r}_i. \tag{3.5}$$

In these new variables, the Hamiltonian is given by

$$\begin{cases} K_i = \dfrac{p_i^2}{2\beta_i} + \dfrac{g_i^2}{2\beta_i r_i^2} - \dfrac{\mu_i\beta_i}{r_i} \\ H_{int} = \dfrac{\mu_2\beta_2}{r_2} - \dfrac{\mu_2\beta_2}{\sqrt{\Delta_1(r_1, r_2, v_2 - v_1)}} - \dfrac{\mu}{\sqrt{\Delta_0(r_1, r_2, v_2 - v_1)}} \end{cases} \tag{3.6}$$

where

$$\begin{cases} \Delta_0(r_1, r_2, w) := r_2^2 + \delta_0^2 r_1^2 + 2\delta_0 r_1 r_2 \cos w \\ \Delta_1(r_1, r_2, w) := r_2^2 + \delta_1^2 r_1^2 + 2\delta_1 r_1 r_2 \cos w \end{cases}$$

The K_i in (3.6) are the Hamiltonians of two independent Kepler problems, in the same form as in (2.1). We notice that the Hamiltonian does not depend on v_1 and v_2 but on $v_1 - v_2$, so we set

$$w := v_1 - v_2, \quad g := g_1$$

and change variables to $(r_1, r_2, w; p_1, p_2, g)$, therefore

$$\begin{cases} K_1 = \dfrac{p_1^2}{2\beta_1} + \dfrac{g^2}{2\beta_1 r_1^2} - \dfrac{\mu_1\beta_1}{r_1} \\ K_2 = \dfrac{p_2^2}{2\beta_2} + \dfrac{(G-g)^2}{2\beta_2 r_2^2} - \dfrac{\mu_2\beta_2}{r_2} \\ H_{int} = \dfrac{\mu_2\beta_2}{r_2} - \dfrac{\mu_2\beta_2}{\sqrt{\Delta_1(r_1, r_2, w)}} - \dfrac{\mu}{\sqrt{\Delta_0(r_1, r_2, w)}} \end{cases} \tag{3.7}$$

where G is the total angular momentum $g_1 + g_2$. The system is thus reduced to three degrees of freedom.

By applying a Legendre transform, we find the system's Lagrangian

$$L(r_1, r_2, w; \dot{r}_1, \dot{r}_2, \dot{w}) = L_1 + L_2 + L_{int} \tag{3.8}$$

where

$$\begin{cases} L_1 := \dfrac{1}{2}\beta_1\dot{r}_1^2 + \dfrac{\mu_1\beta_1}{r_1} + \dfrac{1}{2}\beta_1 r_1^2 \xi^2(r_1, r_2, \dot{w}) \\ L_2 := \dfrac{1}{2}\beta_2\dot{r}_2^2 + \dfrac{\mu_2\beta_2}{r_2} + \chi(r_1, r_2, \dot{w})\left(\beta_1 r_1^2\xi(r_1, r_2, \dot{w}) - \dfrac{1}{2}\beta_2 r_2^2\chi(r_1, r_2, \dot{w})\right) \\ L_{int} := -\dfrac{\mu_2\beta_2}{r_2} + \dfrac{\mu_2\beta_2}{\sqrt{\Delta_1(r_1, r_2, w)}} + \dfrac{\mu}{\sqrt{\Delta_0(r_1, r_2, w)}} \end{cases}$$

and

$$\begin{cases} \xi(r_1, r_2, \dot{w}) := \dfrac{\dot{w}\beta_2 r_2^2 + G}{\beta_1 r_1^2 + \beta_2 r_2^2} = \dot{v}_1 \\ \chi(r_1, r_2, \dot{w}) := \dfrac{\dot{w}\beta_1 r_1^2 - G}{\beta_1 r_1^2 + \beta_2 r_2^2} = \dot{v}_2 \end{cases}$$

3.2. Setting up the Newton algorithm

Following the procedure detailed in section 2, we set up a Newton algorithm to compute the extremum of the action. We express r_1, r_2 and w as generalized Fourier series in three dimensions: if x is one of r_1, r_2 or w,

$$x(\Phi) = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} \sum_{k_3=-\infty}^{\infty} x_{k_1,k_2,k_3} e^{i(k_1\phi_1+k_2\phi_2+k_3\phi_3)}$$

which we rewrite in a more compact form

$$x(\Phi) = \sum_{\mathbf{k}\in\mathbb{Z}^3} x_{\mathbf{k}} e^{i\mathbf{k}\cdot\Phi}. \tag{3.9}$$

Here the $x_{\mathbf{k}}$ are complex, since we cannot suppose that the initial velocities vanish. We however impose that $x(\Phi)$ is real:

$$\mathcal{R}e(x_{-\mathbf{k}}) = \mathcal{R}e(x_{\mathbf{k}}) \text{ and } \mathcal{I}m(x_{-\mathbf{k}}) = -\mathcal{I}m(x_{\mathbf{k}}). \tag{3.10}$$

Thus we must only consider the $\mathbf{k}$ in the set

$$\mathbb{Z}^{3+} := (\mathbb{N}^* \times \mathbb{Z} \times \mathbb{Z}) \cup (\{0\} \times \mathbb{N}^* \times \mathbb{Z}) \cup (\{0\} \times \{0\} \times \mathbb{N})$$

where $\mathbb{N}^* = \mathbb{N} \setminus \{0\}$. We truncate the Fourier series at an order $k_m \in \mathbb{N} \setminus \{0\}$. We define

$$\mathfrak{K} := \{(k_1, k_2, k_3) \in \mathbb{Z}^{3+},\ |k_i| \leqslant k_m\}$$

and rewrite (3.9) as

$$x(\Phi) = \frac{1}{2} \sum_{\mathbf{k}\in\mathfrak{K}} \mathcal{R}e(x_{\mathbf{k}}) \left(e^{i\mathbf{k}\cdot\Phi} + e^{-i\mathbf{k}\cdot\Phi}\right) + i\ \mathcal{I}m(x_{\mathbf{k}}) \left(e^{i\mathbf{k}\cdot\Phi} - e^{-i\mathbf{k}\cdot\Phi}\right). \tag{3.11}$$

The number of elements in $\mathfrak{K}$ is

$$k_m(2k_m+1)^2 + k_m(2k_m+1) + k_m + 1. \tag{3.12}$$

The Newton algorithm is defined as in (2.10), with a slight modification coming from the fact that $x_{\mathbf{k}}$ is complex, which prevents us from deriving S_ω with respect to $x_{\mathbf{k}}$, instead we derive S_ω with respect to the real and imaginary parts of $x_{\mathbf{k}}$.

$$x_{\mathbf{k}}^{(n+1)} - x_{\mathbf{k}}^{(n)} = -\left(D^2 S(r_{\mathbf{k}}^{1,(n)}, r_{\mathbf{k}}^{2,(n)}, w_{\mathbf{k}}^{(n)})\right)^{-1} \partial S_\omega(r_{\mathbf{k}}^{1,(n)}, r_{\mathbf{k}}^{2,(n)}, w_{\mathbf{k}}^{(n)}). \tag{3.13}$$

where if we denote the real and imaginary parts of $x_{\mathbf{k}}$ respectively by $a_{\mathbf{k}}$ and by $b_{\mathbf{k}}$, ∂S_ω is made of three vectors of the form

$$\begin{pmatrix} \dfrac{\partial S_\omega}{\partial a_{\mathbf{k}}} \\ \dfrac{\partial S_\omega}{\partial b_{\mathbf{k}}} \end{pmatrix}$$

and $D^2 S_\omega$ is made of 9 blocks of the form

$$\begin{pmatrix} \dfrac{\partial^2 S_\omega}{\partial a_{\mathbf{k}} \partial a_{\mathbf{k}}} & \dfrac{\partial^2 S_\omega}{\partial a_{\mathbf{k}} \partial b_{\mathbf{k}}} \\ \dfrac{\partial^2 S_\omega}{\partial b_{\mathbf{k}} \partial a_{\mathbf{k}}} & \dfrac{\partial^2 S_\omega}{\partial b_{\mathbf{k}} \partial b_{\mathbf{k}}} \end{pmatrix}.$$

We compute the derivatives of the action S_ω. Their expression is rather long, so it is deferred to appendix A5.

In the preceding discussion, we considered the variables r_1, r_2 and w as Fourier series. However, the fact that w is an angle induces some technical problems, which are discussed in appendix A3.

4. Alternative numerical algorithms

The Newton algorithm described in sections 2 and 3 requires the Hessian matrix of the action to be inverted. Using (3.12), we find that the number of lines and columns of this matrix is

$$N := 6(k_m(2k_m+1)^2 + k_m(2k_m+1) + k_m + 1) \tag{4.1}$$

which for $k_m = 8$ is equal to 14 742. Thus the total number of entries would be 217 326 562. This makes the computations long, and requires prohibitive amounts of memory: since each entry in the matrix uses 64 bits of memory, the matrix will require over 1.7 GB. We will therefore study alternative methods to find the extremum of the action: the *conjugate gradient* method, which requires very little memory but long computation times, and was found to be numerically unstable; and the *quasi-Newton* method, which only requires the Hessian matrix to be stored once but not inverted. Neither algorithm converges for the three body problem. We will discuss the possible reasons for this in section 5.

4.1. The conjugate gradient algorithm

Conceptually, the Newton algorithms performs two tasks: it finds a direction in which to look for the extremum, and determines how big a step to take. The direction is that of the vector

$$-\left(D^2 S_\omega\left(x^{(n)}\right)\right)^{-1} \cdot \partial_k S_\omega\left(x^{(n)}\right)$$

from (2.10), and the size of the step is its norm.

A more naive way of proceeding, which would not require $D^2 S_\omega^{-1}$ to be computed, would be to pick the directions from an arbitrary basis of $\mathbb{R}^N$ and use a method to

extremalize S_ω in one direction after another. However, if the directions are not picked carefully, there is no reason why extremalizing in one of them would not make the solution move away from the extremum in the others. Thus the algorithm would have to be repeated many times before being able to converge. The conjugate gradient method gives a set of directions that are such that an extremalization procedure along one of the directions will not affect the others. Such a set is called a set of *conjugate directions*. The ideas expressed in this section were inspired by [NR], chapter 10, which is in turn based on works by R. Fletcher and C.M. Reeves.

Conceptually, the conjugate gradient method solves an equation of the form

$$Ax = b. \tag{4.2}$$

In this case, A is the Hessian D^2S_ω, x is a vector made of the Fourier coefficients of r_1, r_2 and w, and b is the gradient $-\partial S_\omega$. It introduces a vector h, which will determine the direction in which to search for the solution of (4.2), and an auxiliary vector g. The algorithm is defined by

$$\begin{cases} x_0 \text{ arbitrary} \\ h_0 = g_0 = Ax_0 - b \\ x_{i+1} = x_i - \lambda_i h_i \\ \lambda_i := \dfrac{g_i \cdot h_i}{h_i \cdot Ah_i} \\ g_{i+1} = g_i - \lambda_i A h_i \\ h_{i+1} = g_{i+1} + \dfrac{g_{i+1} \cdot g_{i+1}}{g_i \cdot g_i} h_i. \end{cases} \tag{4.3}$$

It verifies the following lemma: $\forall j < i$,

$$g_i \cdot g_j = 0 \quad h_i \cdot Ah_j = 0 \quad g_i \cdot h_j = 0 \tag{4.4}$$

which implies that the N first g's form a basis of $\mathbb{R}^N$, and so do the N first h's. We recall that N is the size of ∂S. One can then prove that one of the x_i for $i \leqslant N$ verifies (4.2). For details on this result and on the lemma, see appendix A4.

As it is expressed here, the conjugate gradient algorithm requires the computation of the Hessian D^2S_ω. However, one can easily verify that if S_ω is a quadratic function, then (4.3) is equivalent to

$$\begin{cases} x_0 \text{ arbitrary} \\ h_0 = g_0 = \partial S_\omega(x_0) \\ \lambda_i \text{ is the extremum of } \lambda \mapsto S_\omega(x_i - \lambda h_i) \\ x_{i+1} = x_i - \lambda_i h_i \\ g_{i+1} = \partial S_\omega(x_{i+1}) \\ h_{i+1} = g_{i+1} + \dfrac{g_{i+1} \cdot g_{i+1}}{g_i \cdot g_i} h_i. \end{cases} \tag{4.5}$$

If S_ω is not quadratic, it can be approximated by a quadratic function using its Taylor expansion, and thus the algorithm (4.5) would converge in a number of steps of the order of $N \ln N$.

The algorithm (4.5) requires the computation of the extremum of the action in a given direction. Since its Hessian is neither positive nor negative definite, we cannot predict whether this extremum is a maximum or a minimum. We can therefore not use a minimization or a maximization algorithm, but must instead search for the zero of the function

$$f_i : \lambda \longmapsto h_i \cdot \partial S_\omega(x_i - \lambda h_i).$$

To implement this, we use the Van Wijngaarden-Dekker-Brent method (see [NR], chapter 9, and program P2), which requires a few evaluations of f_i. Every evaluation requires a computation time of the order of N, so the entire algorithm's execution time is of the order of $N^2 \ln N$.

The algorithm (4.5) only requires a few vectors to be stored in memory.

On the computer we used (see appendix A7 for detailed specifications), a step would take around 45 seconds for $k_m = 8$. If we performed N iterations, the algorithm would take over a week to be completed.

However, our implementation diverges after around 30 iterations. In fact, we found that the exact conjugate gradient algorithm (4.3) is numerically unstable for large matrices. To show this, we computed a random $N \times N$ symmetric matrix A with elements between -1 and 1, and ran the algorithm to find the solution of

$$Ax = b$$

for some random b. We found that if $N = 20$, the error

$$\frac{1}{N}\|Ax - b\|$$

is under 10^{-26}, but if $N = 100$, the error is over 10^{-2}. The code that gave us these results is provided in program P5

We shall therefore implement a different algorithm to find the extremum of S_ω, that uses more memory, but is faster and numerically stable: the *quasi-Newton* method.

4.2. The quasi-Newton algorithm

The *quasi-Newton* algorithm, also called the *variable metric* method, is an algorithm to find an extremum of a function (in contrast with the Newton method, which finds a zero of a function, which we have used to find an extremum). It has two variants: the Davidon-Fletcher-Powell and the Broyden-Fletcher-Goldfarb-Shanno methods. We shall implement the latter, following [NR], chapter 10.

The idea of the quasi-Newton method is that the inverse of the Hessian of the action $D^2 S_\omega$ does not have to be computed exactly to make the Newton method converge

quadratically. Instead, we approach it by a matrix H. The algorithm is even more crafty: the matrix H is re-computed at each step, which has two advantages: each iteration is as effective as possible, and we may start the algorithm with a crude approximation of $\left(D^2 S_\omega\right)^{-1}$ without it affecting the entire iteration.

It is defined by

$$\begin{cases} x_0 \text{ arbitrary} \\ H_0 \text{ symmetric} \\ \lambda_i \text{ is the extremum of } \lambda \mapsto S_\omega(x_i - \lambda H_i \partial S_\omega(x_i)) \\ x_{i+1} = x_i - \lambda_i H_i \partial S_\omega(x_i) \\ \delta_i := x_{i+1} - x_i \\ s_i := (\partial S_\omega(x_{i+1}) - \partial S_\omega(x_i)) \\ H_{i+1} = H_i + \dfrac{\delta_i \otimes \delta_i}{\delta_i \cdot s_i}\left(1 + \dfrac{s_i \cdot H_i s_i}{\delta_i \cdot s_i}\right) - \left(H_i \dfrac{s_i \otimes \delta_i}{\delta_i \cdot s_i} + \dfrac{\delta_i \otimes (H_i s_i)}{\delta_i \cdot s_i}\right) \end{cases} \tag{4.6}$$

where $\otimes$ denotes the exterior product

$$(x \otimes y)_{i,j} := x_i y_j.$$

E. Polak gives a proof of the super-linear convergence of (4.6) in [Po71]. One can see that (4.6) does in fact approach $D^2 S_\omega$ since the H_i verify

$$\partial S_\omega(x_{i+1}) - \partial S_\omega(x_i) = H_{i+1}^{-1}\left(x_{i+1} - x_i\right). \tag{4.7}$$

How do we choose H_0? The algorithm will converge faster if H_0 is close to $D^2 S_\omega^{-1}(x_0)$. Since we cannot invert $D^2 S$ explicitly, we approach it by neglecting all the terms that come from $\langle\cdot\rangle_{\mathbf{k}}$ with $\mathbf{k} \neq \mathbf{0}$. This reduces $D^2 S_\omega$ to a matrix made of 36 diagonal blocks, which we can invert using the analytical formula for the inverse of a 6×6 matrix. This formula was computed using Mathematica, and is over 10 000 lines long.

The quasi-Newton algorithm requires an $N \times N$ matrix to be stored in memory, but not inverted. Its super-linear convergence implies it takes few steps to produce an extremum of the action. In our implementation, each step takes around 2 minutes.

4.3. Implementation of the quasi-Newton algorithm

We ran the quasi-Newton algorithm for $k_m = 8$, with realistic values for the masses and eccentricities of the Sun-Jupiter-Saturn system. The frequencies we imposed were provided by J. Laskar, and were computed using a *frequency map analysis*. This technique, developed by J. Laskar [La99], consists in performing a numerical integration of the equations of motion over a duration T, and estimating the frequencies from it.

The rate of convergence of the frequencies as a function of T is proportional to T^{-4}, which makes it very efficient. The values we took for the constants and frequencies are given in the table below. The masses are expressed in multiples of Saturn's mass, and the frequencies in rad $\cdot$ year^{-1}.

m_0	3497.89	ω_1	0.529695
m_1	3.33976	ω_2	0.213265
m_2	1	ω_3	-0.000114735
e_1	0.0481470	e_2	0.0538197

The conclusion of the numerical tests is that the algorithm diverges. The code is given in program P3. We shall now discuss the reasons why the numerical analysis may have failed.

5. Perspectives

5.1. The degeneracy problem

A possible reason for the fact that the algorithm does not converge is that there might not be any continuous extremum of the action S_ω. Its existence is ensured by theorem 1.3, however, the theorem is only applicable if the system can be formulated as the perturbation of a *non-degenerate* integrable system, in the sense that the unperturbed system expressed in action-angle variables has a non-degenerate Lagrangian. The three body problem is close to two non-interacting Kepler problems, however, using equation (2.4), one can easily see that Kepler problems are degenerate. Therefore theorem 1.3 can not be applied as such.

In their effort to compute Kolmogorov's normal form for the three body problem, U. Locatelli & A. Giorgilli [LG05] faced this very problem. In the Hamiltonian formalism though, there is a way around the problems created by the degeneracy of the unperturbed theory in the case of the three body problem, developed by V.I. Arnol'd [Ar63b]. The idea expressed in [Ar63b] can be sketched out roughly in the following way. In action angle variables, which we denote by $(M_1, M_2, \varpi; \Lambda_1, \Lambda_2, g)$, the Hamiltonian can be written as

$$H(M_1, M_2, \varpi; \Lambda_1, \Lambda_2, g) = H_0(\Lambda_1, \Lambda_2) + \epsilon H_1(M_1, M_2, \varpi; \Lambda_1, \Lambda_2, g) \tag{5.1}$$

where H_0 describes the non-interacting system and H_1 the perturbation. We notice that $\dot{\varpi}$ is of the order of ϵ, whereas $\dot{M}_1$ and $\dot{M}_2$ are macroscopic. The different angles therefore play very different roles: M_1 and M_2 will evolve *quickly*, whereas ϖ will be *slow*. To lift the degeneracy of H_0, we separate H_1 into a *secular* part(from the latin "sæclum", meaning "century")

$$\overline{H}_1(\varpi; \Lambda_1, \Lambda_2, g) := \oint dM_1 dM_2 \; H_1(M_1, M_2, \varpi; \Lambda_1, \Lambda_2, g)$$

and a *fast* part

$$\widetilde{H}_1(M_1, M_2, \varpi; \Lambda_1, \Lambda_2, g) := H_1(M_1, M_2, \varpi; \Lambda_1, \Lambda_2, g) - \overline{H}_1(\varpi; \Lambda_1, \Lambda_2, g).$$

Using the fact that the rate of variation of ϖ is of the order of ϵ, one can prove that there exists a canonical change of variables such that Hamiltonian (5.1) can be rewritten as

$$H(M_1', M_2', \varpi'; \Lambda_1', \Lambda_2', g') = H_0(\Lambda_1', \Lambda_2') + \epsilon\overline{H}_1(\Lambda_1', \Lambda_2', g) + \epsilon^2 H_2(M_1', M_2', \varpi; \Lambda_1', \Lambda_2', g). \tag{5.2}$$

The KAM theorem can then be applied to (5.2), where the unperturbed motion is given by H_0 and the secular part of H_1.

The question from this perspective is whether a similar manipulation can be performed in Lagrangian formalism to adapt theorem 1.3 to the case of the three body problem. The fact that our algorithm did not converge might be an indication that the variational principle with the action S_ω defined in (1.9) does not give a result if the system is too close to being degenerate, but a more thorough study of theorem 1.3 could reveal another expression for the action that would yield an extremum for the three body problem.

The analysis of this question is beyond the scope of the present work. We will limit ourselves to a first step, that consists in checking that our algorithm works for a simple non-integrable systems, that is close to an integrable non-degenerate system.

5.2. Simple model

We applied the algorithm to a simpler model defined by the Hamiltonian

$$H(\phi_1, \phi_2; I_1, I_2) = \frac{I_1^2}{2} + \frac{I_2^2}{2} + \epsilon(\cos(\phi_1 + \phi_2) + \cos(\phi_1 - \phi_2)). \tag{5.3}$$

This Hamiltonian is not integrable if $\epsilon \neq 0$, and the unperturbed model

$$H_0(I_1, I_2) = \frac{I_1^2}{2} + \frac{I_2^2}{2}$$

is non-degenerate.

The first results given by our algorithm are positive. We found invariant tori for various frequencies and values of ϵ. The system should be studied in more detail to see whether the algorithm is generally reliable.

For example, we studied the system with $\epsilon = 0.1$, $k_m = 16$ and $\omega = (1, 123/200)$, and found a solution that verifies the Euler-Lagrange equation (1.6) up to a precision of 10^{-21}. Furthermore, we compared the computed solution to one found using a *traditional* integration of the equations of motion, and found that the frequencies coincide

up to a precision of 10^{-10}, as well as the Fourier coefficients. The results and a precise comparison with the numerical integration of the equations of motion is given in appendix A6.

The next step would be to study the role of the degeneracy by introducing a parameter α in (5.3)

$$H(\phi_1, \phi_2; I_1, I_2) = \frac{I_1^2}{2} + \alpha \frac{I_2^2}{2} + \epsilon(\cos(\phi_1 + \phi_2) + \cos(\phi_1 - \phi_2)). \tag{5.4}$$

and see what happens when α goes to 0.

Conclusion

The main result of this work is that Percival's variational principle (1.9) cannot be applied as such for the Sun-Jupiter-Saturn system. Indeed, it seems from the algorithm we have detailed above, that the action (1.9) has no continuous extremum. This may come from the fact that the perturbation is too large, but previous numerical computations have shown that on the time scales we are investigating (a few times $1/\omega_3$), the motion is in fact regular. Furthermore, reducing the masses by a factor of 1 000, as was done in [LG05], did not enable the algorithm to converge. Instead, I believe the problem comes from the fact that the system is too close to the Kepler problem, which is degenerate, and theorem 1.3 only proves the existence of the extremum of S_ω if the system in question is close to an integrable non-degenerate system. It is thus possible that the variational problem (1.9) should be revised for close to degenerate systems. This very interesting question is, to my knowledge, open.

In the Hamiltonian formulation of the KAM theorem, the theorem can be applied for some systems that are close to degenerate: [Ar63b] shows a way of making the perturbation theory more precise, effectively changing the unperturbed Hamiltonian to a non-degenerate one; in [CC97], the authors provide an alternative Hamiltonian which yields some of the same trajectories, and which is still close to being degenerate, but no longer close to being *iso-energetically degenerate*, which makes the KAM theorem work (using a later formalism than the one presented in section 1, see [Ar88]). These results give good reason to believe that the problem arising from the fact that the system is close to being degenerate can be overcome.

The extremalization procedure yields promising results for the simple system (5.3). It provides numerical solutions of the problem in a way that is fundamentally different from many other procedures: instead of fixing the energy of a torus, as one usually does, we fix its frequencies. If a continuous invariant torus with the desired frequency exists anywhere in phase space, the extremalization algorithm finds it, regardless of its energy, without having to fix any of its points.

These points should be further investigated.

Appendices

A1. Diophantine condition

We prove that the set of diophantine vectors has full measure in $\mathbb{R}^n$.

Definition A1.1 : A vector $\omega \in \mathbb{R}^n$ is said to be *diophantine* if there exists $c > 0$ and $\eta > 0$ such that

$$\forall k \in \mathbb{Z}^n \setminus \{0\}, \quad |k \cdot \omega| \geqslant \frac{c}{\|k\|^\eta} \tag{A1.1}$$

Lemma A1.1 (second Borel-Cantelli lemma): If a sequence (E_j) of Lebesgue measurable subsets of a compact set in $\mathbb{R}^n$ is such that

$$\sum_j \mathcal{L}(E_j) = \infty$$

where $\mathcal{L}$ denotes the Lebesgue measure, then there is a sequence (F_j) of translates

$$F_j = E_j + x_j$$

such that

$$\limsup F_j := \bigcap_{n=1}^{\infty} \bigcup_{k=n}^{\infty} F_k$$

covers $\mathbb{R}^n$ apart from a set of measure 0.

Property A1.1 : Let $\mathcal{D}_{\eta,c}$ be the set of diophantine vectors with constants η and c as defined in the definition above. If $\eta > n-1$, then the Lebesgue measure of $\mathbb{R}^n \setminus \mathcal{D}_{\eta,c}$ is 0.

Proof (from [Ga04]): Let $r > 0$ and $\mathcal{B}_r$ be the open ball of radius r. Let $\mathcal{L}$ denote the Lebesque measure of a set. We first prove that

$$\mathcal{L}(\mathbb{R}^n \setminus \mathcal{D}_{\eta,c} \cap \mathcal{B}_r) \leqslant K_\eta c r^{n-1} \tag{A1.2}$$

for some $K_\eta > 0$ that depends solely on η. Notice that

$$\left\{\omega,\ \exists k \in \mathbb{Z}^n,\ |\omega \cdot k| < \frac{c}{\|k\|^\eta}\right\} \subset \bigcup_{k \in \mathbb{Z}^n} \left\{\omega,\ \ \|\omega\| < \frac{c}{\|k\|^{\eta+1}}\right\} =: \bigcup_{k \in \mathbb{Z}^n} \mathcal{A}_{\eta,k}$$

so

$$\mathcal{L}(\mathbb{R}^n \setminus \mathcal{D}_{\eta,c} \cap \mathcal{B}_r) \leqslant \sum_{k \in \mathbb{Z}^n} \mathcal{L}(\mathcal{A}_{\eta,k})$$

For each $k \in \mathbb{Z}^n$, there are two cases:

$* \; c\|k\|^{-\eta-1} < r$, in which case

$$\mathcal{L}(\mathcal{A}_{\eta,k}) = S_n c^n \|k\|^{-n(\eta+1)} < S_n c \|k\|^{-\eta-1} r^{n-1}$$

where S_n is the volume of the unit sphere in n dimensions.

$* \; c\|k\|^{-\eta-1} \geqslant r$ in which case

$$\mathcal{L}(\mathcal{A}_{\eta,k}) = S_n r^n \leqslant S_n c \|k\|^{-\eta-1} r^{n-1}$$

Therefore

$$\mathcal{L}(\mathbb{R}^n \setminus \mathcal{D}_{\eta,c} \cap \mathcal{B}_r) \leqslant S_n c r^{n-1} \sum_{k \in \mathbb{Z}^n} \frac{1}{\|k\|^{\eta+1}}$$

and since $\eta > n-1$,

$$\sum_{k \in \mathbb{Z}^n} \frac{1}{\|k\|^{\eta+1}}$$

converges. This proves (A1.2). Thus

$$\mathcal{L}(\mathcal{D}_{\eta,c} \cap \mathcal{B}_r) \geqslant S_n r^n - K_\eta c r^{n-1}$$

So if we pick a sequence of r going to infinity, we get a sequence of (E_n) to which we may apply the Borel-Cantelli lemma, which implies that

$$\mathcal{L}(\mathbb{R}^n \setminus \mathcal{D}_{\eta,c}) = 0$$

A2. The KAM theorem

In this appendix we give the statement and a sketch of the proof of Kolmogorov's version of the KAM theorem.

Theorem A2.1 : Consider a system whose Hamiltonian is given by

$$H(\mathbf{q};\mathbf{p}) = H_0(\mathbf{q};\mathbf{p}) + \epsilon H_1(\mathbf{q};\mathbf{p})$$

such that H_0 is integrable. We suppose that H is analytic on the strip of the complex plane defined by $\mathcal{I}m(q_i) \leqslant \rho$ and that the matrix $\mathcal{H}$ defined by

$$\mathcal{H}_{\alpha,\beta}(\mathbf{q}) = \frac{\partial^2 H}{\partial p_\alpha \partial p_\beta}(\mathbf{q};0)$$

satisfies

$$\det(\mathcal{H}(\mathbf{q})) \neq 0.$$

The latter condition is called the *non-degeneracy condition*. Take $(\mathbf{q}_0;\mathbf{p}_0)$ a point in phase space. Let ω be the frequency vector of the trajectory computed for $\epsilon = 0$, that starts at $(\mathbf{q}_0;\mathbf{p}_0)$. If ω is *diophantine* with parameters η and c, then there exists $\epsilon_0 > 0$, which may depend on η and c, such that for any $\epsilon < \epsilon_0$, there exists a trajectory starting from a point in the neighborhood of $(\mathbf{q}_0;\mathbf{p}_0)$ that is quasi-periodic, with a frequency vector equal to ω.

Idea of the proof [Ko54]: We suppose $(\mathbf{q};\mathbf{p})$ are action-angle variables of H_0. The unperturbed trajectory has a constant momentum $\mathbf{I}$, but since the tranlation of $\mathbf{p}$ by a constant vector is a canonical transformation, we can suppose $\mathbf{I} = 0$. Thus we may write H as

$$H(\mathbf{q};\mathbf{p}) = m + \omega\cdot\mathbf{p} + \frac{1}{2}\mathbf{p}^T\mathcal{H}(\mathbf{q})\mathbf{p} + \epsilon\left(A(\mathbf{q}) + \mathbf{B}(\mathbf{q})\cdot\mathbf{p}\right) + O(\|\mathbf{p}\|^3) \tag{A2.1}$$

where $m \in \mathbb{R}$, $\mathbf{B}(\mathbf{q}) \in \mathbb{R}^n$ and $\mathbf{p}^T$ is the transpose of the column vector $\mathbf{p}$. The essence of the proof is to find a canonical transform to the variables $(\mathbf{q}';\mathbf{p}')$ such that (A2.1) becomes

$$H(\mathbf{q}';\mathbf{p}') = M(\epsilon) + \omega\cdot\mathbf{p}' + O(\|\mathbf{p}'\|^2) \tag{A2.2}$$

This would prove the theorem since the trajectory of the perturbed system starting from $(\mathbf{q}';\mathbf{p}' = 0)$ remains at $\mathbf{p}' = 0$, thus staying on a torus at the same frequency ω. To find such a change of variables, we proceed by steps, eliminating the unwanted terms at each order in ϵ. We detail the first step: we perform a canonical transform from the variables $(\mathbf{q};\mathbf{p})$ to $(\mathbf{Q},\mathbf{P})$. To construct the canonical change of variables, we use a *generating function*

$$S(\mathbf{q},\mathbf{P}) = \mathbf{q}\cdot(\epsilon\xi + \mathbf{P}) + \epsilon X(\mathbf{q}) + \epsilon\mathbf{Y}(\mathbf{q})\cdot\mathbf{P} \tag{A2.3}$$

where $\mathbf{Y}$ and X are arbitrary $\mathcal{C}^1$ functions and ξ is an arbitrary vector. From (A2.3) we find

$$\begin{cases} \mathbf{Q} = \dfrac{\partial S}{\partial \mathbf{P}} = \mathbf{q} + \epsilon\mathbf{Y}(\mathbf{q}) \\ \mathbf{p} = \dfrac{\partial S}{\partial \mathbf{q}} = \mathbf{P} + \epsilon\left(\xi + \dfrac{\partial X}{\partial \mathbf{q}}(\mathbf{q}) + \displaystyle\sum_i P_i \frac{\partial Y_i}{\partial \mathbf{q}}(\mathbf{q})\right) \end{cases} \tag{A2.4}$$

We require that $H(\mathbf{Q};\mathbf{P})$ take the form

$$H(\mathbf{Q};\mathbf{P}) = m + \epsilon\zeta + \omega\mathbf{P} + O\left(\|\mathbf{P}\|^2,\epsilon^2\right). \tag{A2.5}$$

We have

$$\begin{aligned} H(\mathbf{Q};\mathbf{P}) = m + \omega\cdot\mathbf{P} &+ \epsilon\left(\omega\cdot\xi + \omega\cdot\frac{\partial X}{\partial \mathbf{q}}(\mathbf{q}) + \sum_i P_i\omega\cdot\frac{\partial Y_i}{\partial \mathbf{q}}(\mathbf{q})\right) \\ &+\epsilon\mathbf{P}^T\mathcal{H}(\mathbf{q})\left(\xi + \frac{\partial X}{\partial \mathbf{q}}(\mathbf{q})\right) + \epsilon\left(A(\mathbf{q}) + \mathbf{B}(\mathbf{q})\cdot\mathbf{P}\right) + O\left(\|\mathbf{P}\|^2,\epsilon^2\right) \end{aligned} \tag{A2.6}$$

thus to get (A2.5), we impose

$$\begin{cases} \omega \cdot \left(\xi + \dfrac{\partial X}{\partial \mathbf{q}}(\mathbf{q})\right) + A(\mathbf{q}) = \zeta \\ \displaystyle\sum_i \omega_i \frac{\partial \mathbf{Y}}{\partial q_i}(\mathbf{q}) + \mathcal{H}(\mathbf{q}) \left(\xi + \frac{\partial X}{\partial \mathbf{q}}(\mathbf{q})\right) + \mathbf{B}(\mathbf{q}) = 0. \end{cases} \tag{A2.7}$$

We define

$$\mathbf{Z}(\mathbf{q}) := \mathcal{H} \frac{\partial X}{\partial \mathbf{q}}(\mathbf{q}).$$

We suppose X and $\mathbf{Y}$ are decomposable into Fourier series, and define

$$\begin{cases} \mathcal{H}(\mathbf{q}) = \displaystyle\sum_{\mathbf{k} \in \mathbb{Z}^n} h_{\mathbf{k}} e^{i\mathbf{k}\cdot\mathbf{q}} \\ A(\mathbf{q}) = \displaystyle\sum_{\mathbf{k} \in \mathbb{Z}^n} a_{\mathbf{k}} e^{i\mathbf{k}\cdot\mathbf{q}} \\ \mathbf{B}(\mathbf{q}) = \displaystyle\sum_{\mathbf{k} \in \mathbb{Z}^n} \mathbf{b}_{\mathbf{k}} e^{i\mathbf{k}\cdot\mathbf{q}} \\ X(\mathbf{q}) = \displaystyle\sum_{\mathbf{k} \in \mathbb{Z}^n} x_{\mathbf{k}} e^{i\mathbf{k}\cdot\mathbf{q}} \\ \mathbf{Y}(\mathbf{q}) = \displaystyle\sum_{\mathbf{k} \in \mathbb{Z}^n} \mathbf{y}_{\mathbf{k}} e^{i\mathbf{k}\cdot\mathbf{q}} \\ \mathbf{Z}(\mathbf{q}) = \displaystyle\sum_{\mathbf{k} \in \mathbb{Z}^n} \mathbf{z}_{\mathbf{k}} e^{i\mathbf{k}\cdot\mathbf{q}}. \end{cases}$$

Thus (A2.7) becomes

$$\begin{cases} \omega \cdot \xi + a_0 = \zeta \\ i\mathbf{k} \cdot \omega \; x_{\mathbf{k}} + a_{\mathbf{k}} = 0 \\ h_0 \xi + \mathbf{z}_0 + \mathbf{b}_0 = 0 \\ i\mathbf{k} \cdot \omega \; \mathbf{y}_{\mathbf{k}} + h_{\mathbf{k}} \xi + \mathbf{z}_{\mathbf{k}} + \mathbf{b}_{\mathbf{k}} = 0. \end{cases} \tag{A2.8}$$

We get

$$x_{\mathbf{k}} = i \frac{a_{\mathbf{k}}}{\mathbf{k} \cdot \omega}$$

thus determining X up to a constant and thus $\mathbf{z}_{\mathbf{k}}$ as well as $\mathbf{z}_0$. We can then find

$$\xi = -h_0^{-1}(\mathbf{z}_0 + \mathbf{b}_0)$$

which gives ζ and

$$\mathbf{y}_{\mathbf{k}} = i \frac{h_{\mathbf{k}} \xi + \mathbf{z}_{\mathbf{k}} + \mathbf{b}_{\mathbf{k}}}{\mathbf{k} \cdot \omega}$$

To prove that X and $\mathbf{Y}$ are analytic, we use the following lemma

> **Lemma A2.1** : Let f be a function. If f is analytic on the strip of the complex plane defined by $\mathcal{I}m(x) \leqslant \rho$ then it may be expanded as a Fourier series with Fourier coefficients f_k that decrease exponentially: there exists $K > 0$ such that
>
> $$|f_k| < Ke^{-\rho k}.$$
>
> Conversely, if f is expandable of a Fourier series and its Fourier coefficients f_k verify
>
> $$|f_k| < Ke^{-\rho k}.$$
>
> for some constants $K > 0$ and $\rho > 0$, then f is analytic on the strip of the complex plane defined by $\mathcal{I}m(x) < \rho$.

Thus there exists $K > 0$ such that

$$|a_{\mathbf{k}}| < Ke^{-\rho\|\mathbf{k}\|}$$

which, using the diophantine condition, implies that

$$|x_{\mathbf{k}}| < \frac{K}{c}\|\mathbf{k}\|^{\eta}e^{-\rho\|\mathbf{k}\|}$$

Thus for any $0 < h < p$, X is analytic on strip of the complex plane defined by

$$\mathcal{I}m(x) \leqslant \rho - h.$$

We may proceed similarly for $\mathbf{y_k}$ and $\mathbf{z_k}$.

The terms of higher order in ϵ and $\mathbf{P}$ that we neglected in (A2.6) are

$$\frac{1}{2}\mathbf{P}^T\mathcal{H}(\mathbf{q})\mathbf{P} + \sum_i \mathbf{P}^T\mathcal{H}(\mathbf{q})P_i\frac{\partial Y_i}{\partial \mathbf{q}}(\mathbf{q}) + \epsilon^2\mathbf{B}(\mathbf{q}) \cdot \left(\xi + \frac{\partial X}{\partial \mathbf{q}}(\mathbf{q}) + \sum_i P_i\frac{\partial Y_i}{\partial \mathbf{q}}(\mathbf{q})\right) \tag{A2.9}$$

$$+\epsilon^2\left(\xi + \frac{\partial X}{\partial \mathbf{q}}(\mathbf{q}) + \sum_i P_i\frac{\partial Y_i}{\partial \mathbf{q}}(\mathbf{q})\right)^T \mathcal{H}\left(\xi + \frac{\partial X}{\partial \mathbf{q}}(\mathbf{q}) + \sum_i P_i\frac{\partial Y_i}{\partial \mathbf{q}}(\mathbf{q})\right)$$

as well as terms coming from $O(\|\mathbf{p}\|^3)$. All of these terms are analytic in $\mathbf{q}$ and therefore in $\mathbf{Q}$. Thus H is analytic in $\mathbf{Q}$ and $\mathbf{P}$. Therefore (A2.5) is of the form (A2.1), where ϵ is replaced by ϵ^2 and ρ by $\rho - h$. We can thus apply the same step again, which will yield

$$H(\mathbf{Q};\mathbf{P}) = m + \epsilon\zeta_1 + \epsilon^2\zeta_2 + \omega \cdot \mathbf{P} + O(\|\mathbf{P}\|^2, \epsilon^4) \tag{A2.10}$$

and so on.

To prove the theorem, we would need to prove that $\mathbf{Y}$ and X decrease, thus proving the convergence of the change of variables (A2.4). We would then have to check

that after applying all the changes of variables, the Hamiltonian is still well defined and analytic in a domain such that $\rho > 0$. The details of these questions can be found in [Ar63a].

The questions regarding the convergence of the KAM iteration are non-trivial: in the first step, we added terms not only of order ϵ^2, but of all greater orders as well (because of the dependenc in $\mathbf{q}$ instead of $\mathbf{Q}$ of the neglected terms). Therefore at a given step h, the A and $\mathbf{B}$ and consequently X and $\mathbf{Y}$ will not only depend on the iteration at $h-1$ but on the entire history of the iteration. This is similar to a problem that appears when studying the β-function flow in a renormalization group analysis. In fact there is a common language for treating both of these problems that was developed by G. Gallavotti [Ga94, Ga04]. The problem of the convergence of the KAM iteration is re-expressed as the convergence of a Lindstedt series, that can be represented as a sum over a set of trees, analogous to Feynman diagrams where the propagator is given by the *small divisor*

$$\frac{1}{\omega \cdot \mathbf{k}}.$$

The problem is then solved by performing a re-summation of the trees, in the same way as in renormalization group analyses.

A3. W as a Fourier series

In section 3, we have written r_1, r_2 and w as Fourier series, assuming these quantities are periodic analytic functions. However, since w is an angle, it can not necessarily be written as a Fourier series. Instead, we may consider $W(M) := e^{iw(M)}$ which is a periodic analytic complex function, that is thus expressible as a Fourier series. Furthermore, one notices that L can be expressed using only r_1, r_2 and W. To impose that w is real, on must impose a constraint on the Fourier coefficients of W so that

$$|W| = 1 \Longleftrightarrow \forall k \in \mathbb{N}, \sum_{k'=-\infty}^{\infty} W_{k'} W_{k'-k} = \delta_{k,0}. \tag{A3.1}$$

However, the condition (A3.1) is difficult to impose, in fact, if one only considers a finite number of harmonics W_k, (A3.1) can only be satisfied if one of the W_k is equal to 1 and the others are equal to 0 (see below). We can therefore not use W, instead, we write

$$w(M) = \mathbf{v} \cdot M + \sum_{\mathbf{k} \in \mathbb{Z}^3} w_{\mathbf{k}} e^{i\mathbf{k} \cdot M} \tag{A3.2}$$

for some constant vector $\mathbf{v}$ such that $v_i = n_i \omega_i$ for some integer vector $\mathbf{n}$. The fact that w can be expressed in the form (A3.2) is equivalent to the fact that W is Fourier decomposable.

Since $\mathbf{n}$ is a vector of integers, assuming that w depends continuously on the size of the interaction term L_{int} (which is a consequence of the KAM theorem if ω is

diophantine), and assuming that the interaction term is small enough, **n** is the same with or without interactions. It can be explicitly computed for the Kepler problem, which yields

$$w(M) = \omega \cdot M + \sum_{\mathbf{k}\in\mathbb{Z}^3} w_{\mathbf{k}} e^{i\mathbf{k}\cdot M}. \tag{A3.3}$$

We now study the question of how to impose that $w \in \mathbb{R}$ if we consider $W := e^{iw}$ as a variable. To that end, we shall consider the one dimensional case where W is a function $\mathbb{R} \longrightarrow \mathbb{C}$ such that

$$W(t) = \sum_{k=-k_m}^{k_m} W_k e^{ikt}$$

with $W_k \in \mathbb{R}$ and

$$W(t)W^*(t) = 1 \tag{A3.4}$$

and prove that the only way of imposing (A3.4) is that one of the W_k is equal to 1 and the others are equal to 0. We use the following notation to denote sets of consecutive integers:

$$\{n, \cdots, n+p\} =: [|n, n+p|].$$

Lemma A3.1 : We have

$$W(t)W(t)^* = 1 \Longleftrightarrow \forall k \in [|0, 2k_m|], \sum_{k'=k-k_m}^{k_m} W_{k'} W_{k'-k} = \delta_{k,0}$$

Proof: We compute the k'th Fourier coefficient of WW^*: $\forall k \in \mathbb{N}$, the term in e^{ikt} is

$$\sum_{k'=k-k_m}^{k_m} W_{k'} W_{k'-k}$$

and the term in e^{-ikt} is

$$\sum_{k'=-k_m}^{k_m-k} W_{k'} W_{k'+k}$$

Furthermore

$$\sum_{k'=-k_m}^{k_m-k} W_{k'} W_{k'+k} = \sum_{k'=-k_m+k}^{k_m} W_{k'-k} W_{k'}$$

This proves the lemma.

Theorem A3.1 For $k_m \in \mathbb{N}$, let

$$W(t) := \sum_{k=-k_m}^{k_m} W_k e^{ikt}$$

If $\forall t \in \mathbb{R}$

$$W(t)W(t)^* = 1$$

then $\exists k \in [|-k_m, k_m|]$ such that

$$\begin{cases} W_k \in \{-1, +1\} \\ W_{k'} = 0 \text{ if } k' \neq k \end{cases}$$

Proof: We prove the theorem by induction on k_m. For $k_m = 0$, the statement is obvious. For $k_m \in \mathbb{N}^*$, using the previous lemma, we find that $\forall k \in [|0, 2k_m|]$

$$\sum_{k'=k-k_m}^{k_m} W_{k'} W_{k'-k} = \delta_{k,0}$$

Therefore, by taking this expression for $k = 2k_m$,

$$W_{k_m} W_{-k_m} = 0$$

and for $k = 2k_m - 1$,

$$W_{k_m-1} W_{-k_m} + W_{k_m} W_{-k_m+1} = \delta_{2k_m-1,0}$$

There are three cases:

* $W_{k_m} = 0$ and $W_{k_m-1} = 0$
* $W_{k_m} = 0$ and $W_{-k_m} = 0$
* $W_{-km} = 0$ and $W_{-k_m+1} = 0$

We will only treat the first case in detail, since the other ones can be treated in the same way. We define

$$\tilde{W}^{(k_m-1)}(t) := \sum_{k=-(k_m-1)}^{k_m-1} W_{k-1} e^{ikt}$$

We have

$$W(t) = e^{-it} \tilde{W}^{(k_m-1)}(t)$$

so

$$\tilde{W}^{(k_m-1)}(t) \tilde{W}^{(k_m-1)}(t)^* = 1$$

By using the induction assumption, $\exists k \in [|-k_m, k_m - 2|]$ such that

$$\begin{cases} W_k \in \{-1, +1\} \\ W_{k'} = 0 \text{ if } k' \neq k \end{cases}$$

This equality also holds for W_{k_m-1} and W_{k_m} since they are both equal to 0.

The other cases can be treated in the same way. The theorem is thus proven.

A4. The conjugate gradient method

In this appendix, we prove the conjugate gradient method. We use the following notations: a set of consecutive integers $\{n, \cdots, n+p\}$ is denoted by $[|n, n+p|]$, $\mathcal{M}_n(\mathbb{R})$ is the set of $n \times n$ real matrices, $GL_n(\mathbb{R})$ is the set of invertible $n \times n$ real matrices. Vectors are considered as column vectors, and ^{T} denotes the transposition operator.

The algorithm gives the solution of

$$Ax = b$$

with $A \in \mathcal{M}_n(\mathbb{R})$ symmetric and $b \in \mathbb{R}^n$, in at most n steps.

Lemma A4.1 Let $A \in GL_n(\mathbb{R})$ be a symmetric invertible matrix. Let $g_0 \in \mathbb{R}^n \setminus \{0\}$ and $\forall i \in [|0, n|]$, we define

$$h_0 := g_0$$

$$\lambda_i := \frac{g_i^T h_i}{h_i^T A h_i}$$

$$g_{i+1} := g_i - \lambda_i A h_i$$

$$\gamma_i := \frac{g_{i+1}^T g_{i+1}}{g_i^T g_i}$$

$$h_{i+1} := g_{i+1} + \gamma_i h_i.$$

Then $\forall i \in [|1, n|]$, $\forall j \in [|0, i-1|]$,

$$g_i^T g_j = 0 \quad h_i^T A h_j = 0 \quad g_i^T h_j = 0.$$

Proof: We prove the theorem by induction on i:

• For $i = 1$,

$$\begin{aligned} g_1^T g_0 &= g_0^T g_0 - \lambda_0 h_0^T A^T g_0 \\ &= g_0^T g_0 - \frac{g_0^T h_0}{h_0^T A h_0} g_0^T A h_0 \\ &= 0 \end{aligned}$$

and

$$\begin{aligned} h_1^T A h_0 &= -\frac{1}{\lambda_0} h_1^T (g_1 - g_0) \\ &= -\frac{1}{\lambda_0} (g_1 + \gamma_0 h_0)^T (g_1 - g_0) \\ &= -\frac{1}{\lambda_0} (g_1^T g_1 - \gamma_0 g_0^T g_0) \\ &= 0 \end{aligned}$$

and

$$g_1^T h_0 = g_1^T g_0 = 0.$$

• For a given $i \geqslant 1$,

$$\begin{aligned} g_{i+1}^T g_i &= g_i^T g_i - \lambda_i h_i^T A^T g_i \\ &= g_i^T g_i - \frac{g_i^T h_i}{h_i^T A h_i} g_i^T A h_i \\ &= g_i^T g_i - \frac{g_i^T (g_i + \gamma_{i-1} h_{i-1})}{h_i^T A h_i} (h_i - \gamma_{i-1} h_{i-1})^T A h_i \end{aligned}$$

using

$$g_i^T h_{i-1} = 0, \quad h_{i-1}^T A h_i$$

we find

$$g_{i+1}^T g_i = 0.$$

Then, for $j \in [|1, i-1|]$,

$$\begin{aligned} g_{i+1}^T g_j &= g_i^T g_j - \lambda_i h_i^T A^T g_j \\ &= -\lambda_i g_j^T A h_i \\ &= -\lambda_i (h_j - \gamma_{j-1} h_{j-1})^T A h_i \\ &= 0. \end{aligned}$$

Finally,

$$g_{i+1}^T g_0 = g_i^T g_0 - \lambda_i h_i^T A^T g_0 = -\lambda_i h_i^T A^T h_0$$

and

$$h_i^T A^T h_0 = h_i^T A h_0 = 0$$

since A is symmetric. Furthermore, $\forall j \in [|0, i|]$,

$$h_{i+1}^T A h_j = -\frac{1}{\lambda_j} h_{i+1}^T (g_{j+1} - g_j)$$

and

$$\begin{aligned} h_{i+1}^T g_j &= (g_{i+1} + \gamma_i h_i)^T g_j \\ &= \begin{cases} g_{i+1}^T g_{i+1} \text{ if } j = i+1 \\ \gamma_i h_i^T g_j \text{ if not} \end{cases} \\ &\vdots \\ &= \prod_{k=j}^{i} \gamma_k \; g_j^T g_j \\ &= \frac{g_i^T g_i}{g_j^T g_j} g_j^T g_j \\ &= g_i^T g_i \end{aligned}$$

so

$$h_{i+1}^T A h_j = 0$$

and

$$g_{i+1}^T h_j = (g_i - \lambda_i A h_i)^T h_j.$$

If $j < i$,

$$g_{i+1}^T h_j = 0$$

and

$$g_{i+1}^T h_i = g_i^T h_i - \frac{g_i^T h_i}{h_i^T A h_i} h_i^T A h_i = 0.$$

Comment : The invertibility of A is necessary for λ_i to be well defined, since

$$\text{Vect}\left(\{h_i\}\right) = \text{Vect}\left(\{g_i\}\right) = \mathbb{R}^n$$

Notice that the λ_i are well defined since $h_i \neq 0$ (consequence of the orthogonality of the g_i).

Therefore, the only issue may arise if $h_i = 0$, which, by virtue of the orthogonality of g_i with al the preceding h_i may only occur if $g_i = 0$, in which case the result of the following theorem remains true.

Theorem 5.1 Let $x_0 \in \mathbb{R}^n$, $b \in \mathbb{R}^n$, $A \in \mathcal{M}_n(\mathbb{R})$ symmetric,

$$g_0 := Ax_0 - b$$

and define for $i \in [|0, n-1|]$

$$x_{i+1} := x_i - \lambda_i h_i$$

with

$$h_0 := g_0$$

$$\lambda_i := \frac{g_i^T h_i}{h_i^T A h_i}$$

$$g_{i+1} := g_i - \lambda_i A h_i$$

$$\gamma_i := \frac{g_{i+1}^T g_{i+1}}{g_i^T g_i}$$

$$h_{i+1} := g_{i+1} + \gamma_i h_i.$$

Then $\exists k \in [|0, \text{rank}(A)|]$ such that x_k verifies

$$Ax_k = b$$

Proof:

• If $\text{rank}(A) = n$, we prove by induction that $\forall i \in [|1, n|]$, $\forall j \in [|0, i-1|]$ we have

$$h_j^T A x_i = h_j^T b.$$

If $i = 1$,

$$h_0^T A x_1 = h_0^T A(x_0 - \lambda_0 h_0) = h_o^T (g_0 + b) - g_0^T h_0 = h_0^T b.$$

If $i \geqslant 1$, for $j < i$

$$h_j^T A x_{i+1} = h_j^T A(x_i - \lambda_i h_i) = h_j^T b$$

and

$$h_i^T A x_{i+1} = h_i^T A(x_i - \lambda_i h_i)$$

$$= h_i^T A x_i - g_i^T h_i$$

and a simple induction shows that

$$g_i = Ax_i - b$$

which proves that $\forall i \in [|1, n|]$, $\forall j \in [|0, i-1|]$ we have

$$h_j^T A x_i = h_j^T b.$$

The previous result implies that (taking $i = n$)

$$Ax_n - b = 0$$

since

$$\text{Vect}\left(\{h_i\}\right) = \text{Vect}\left(\{g_i\}\right) = \mathbb{R}^n.$$

• If $\text{rank}(A) < n$, then we change variables and consider a reduced matrix $A' \in GL_{\text{rank}(A)}(\mathbb{R})$ in order to use the previous lemma.
• If, as was mentioned in the comment, the algorithm reaches a point where $g_i = 0$, then

$$Ax_i = b$$

by virtue of $g_i = Ax_i - b$ (see earlier). If not, the algorithm continues until i reaches $\text{rank}(A)$.

A5. Derivatives of the action

In this appendix we give the expression of the derivatives of the action for the three body problem in Hill-Jacobi variables. We split this computation into two parts: we first find the expression of the two first derivatives of the action as a function of the derivatives of the Lagrangian, then we compute the derivatives of the Lagrangian.

A5.1. Derivatives of the action as a function of the Lagrangian

We define

$$\langle\cdot\rangle_{-\mathbf{k}} = \oint d\Phi \; e^{i\mathbf{k}\cdot\Phi}.$$

and recall

$$S_\omega[r_1, r_2, w] := \oint d\Phi \; L(r_1(\Phi), r_2(\Phi), w(\Phi); D_\omega r_1(\Phi), D_\omega r_2(\Phi), D_\omega w(\Phi)) = \langle L\rangle_{\mathbf{0}}\,.$$

If x is any of r_1, r_2 or w, and $a_{\mathbf{k}}$ and $b_{\mathbf{k}}$ are the real and imaginary parts of $x_{\mathbf{k}}$, using the symmetry (3.10), we have for $\mathbf{k} \neq \mathbf{0}$

$$\begin{cases} \dfrac{\partial x(\Phi)}{\partial a_{\mathbf{k}}} = e^{i\mathbf{k}\cdot\Phi} + e^{-i\mathbf{k}\cdot\Phi} \\ \dfrac{\partial x(\Phi)}{\partial b_{\mathbf{k}}} = ie^{i\mathbf{k}\cdot\Phi} - ie^{-i\mathbf{k}\cdot\Phi} \end{cases} \quad \text{and} \quad \begin{cases} \dfrac{\partial x(\Phi)}{\partial a_{\mathbf{0}}} = 1 \\ \dfrac{\partial x(\Phi)}{\partial b_{\mathbf{0}}} = 0 \end{cases}$$

and

$$\begin{cases} \dfrac{\partial D_\omega x(\Phi)}{\partial a_{\mathbf{k}}} = i\omega\cdot\mathbf{k}\,\left(e^{i\mathbf{k}\cdot\Phi} - e^{-i\mathbf{k}\cdot\Phi}\right) \\ \dfrac{\partial D_\omega x(\Phi)}{\partial b_{\mathbf{k}}} = -\omega\cdot\mathbf{k}\,\left(e^{i\mathbf{k}\cdot\Phi} + e^{-i\mathbf{k}\cdot\Phi}\right) \end{cases} \quad \text{and} \quad \begin{cases} \dfrac{\partial D_\omega x(\Phi)}{\partial a_{\mathbf{0}}} = 0 \\ \dfrac{\partial D_\omega x(\Phi)}{\partial b_{\mathbf{0}}} = 0 \end{cases}$$

which implies that

$$\begin{cases} \dfrac{\partial S_\omega}{\partial a_{\mathbf{k}}} = 2\mathcal{R}e\left\langle \dfrac{\partial L}{\partial x}\right\rangle_{\mathbf{k}} + 2\omega\cdot\mathbf{k}\,\mathcal{I}m\left\langle \dfrac{\partial L}{\partial \dot{x}}\right\rangle_{\mathbf{k}} \\ \dfrac{\partial S_\omega}{\partial b_{\mathbf{k}}} = 2\mathcal{I}m\left\langle \dfrac{\partial L}{\partial x}\right\rangle_{\mathbf{k}} - 2\omega\cdot\mathbf{k}\,\mathcal{R}e\left\langle \dfrac{\partial L}{\partial \dot{x}}\right\rangle_{\mathbf{k}} \end{cases} \quad \text{and} \quad \begin{cases} \dfrac{\partial S_\omega}{\partial a_{\mathbf{0}}} = \left\langle \dfrac{\partial L}{\partial x}\right\rangle_{\mathbf{0}} \\ \dfrac{\partial S_\omega}{\partial b_{\mathbf{0}}} = 0 \end{cases} \tag{A5.1}$$

and if x' is any of r_1, r_2 or w and $a'_{\mathbf{k}}$ and $b'_{\mathbf{k}}$ are the real and imaginary parts of $x'_{\mathbf{k}}$, then for any $\mathbf{l}\neq\mathbf{0}$, we have

$$\begin{cases} \dfrac{\partial^2 S_\omega}{\partial a'_{\mathbf{l}}\partial a_{\mathbf{k}}} = 2\mathcal{R}e\left(\left(\left\langle \dfrac{\partial^2 L}{\partial x'\partial x}\right\rangle_{\mathbf{k+l}} + \left\langle \dfrac{\partial^2 L}{\partial x'\partial x}\right\rangle_{\mathbf{k-l}}\right)\right. \\ \qquad\qquad +(\omega\cdot\mathbf{k})(\omega\cdot\mathbf{l})\left(-\left\langle \dfrac{\partial^2 L}{\partial \dot{x}'\partial \dot{x}}\right\rangle_{\mathbf{k+l}} + \left\langle \dfrac{\partial^2 L}{\partial \dot{x}'\partial \dot{x}}\right\rangle_{\mathbf{k-l}}\right)\Big) \\ \qquad +2\mathcal{I}m\left(\omega\cdot\mathbf{k}\left(\left\langle \dfrac{\partial^2 L}{\partial \dot{x}'\partial x}\right\rangle_{\mathbf{k+l}} + \left\langle \dfrac{\partial^2 L}{\partial \dot{x}'\partial x}\right\rangle_{\mathbf{k-l}}\right)\right. \\ \qquad\qquad +\omega\cdot\mathbf{l}\left(\left\langle \dfrac{\partial^2 L}{\partial x'\partial \dot{x}}\right\rangle_{\mathbf{k+l}} - \left\langle \dfrac{\partial^2 L}{\partial x'\partial \dot{x}}\right\rangle_{\mathbf{k-l}}\right)\Big) \\ \dfrac{\partial^2 S_\omega}{\partial b'_{\mathbf{l}}\partial a_{\mathbf{k}}} = 2\mathcal{I}m\left(\left(\left\langle \dfrac{\partial^2 L}{\partial x'\partial x}\right\rangle_{\mathbf{k+l}} - \left\langle \dfrac{\partial^2 L}{\partial x'\partial x}\right\rangle_{\mathbf{k-l}}\right)\right. \\ \qquad\qquad -(\omega\cdot\mathbf{k})(\omega\cdot\mathbf{l})\left(\left\langle \dfrac{\partial^2 L}{\partial \dot{x}'\partial \dot{x}}\right\rangle_{\mathbf{k+l}} + \left\langle \dfrac{\partial^2 L}{\partial \dot{x}'\partial \dot{x}}\right\rangle_{\mathbf{k-l}}\right)\Big) \\ \qquad +2\mathcal{R}e\left(\omega\cdot\mathbf{k}\left(-\left\langle \dfrac{\partial^2 L}{\partial \dot{x}'\partial x}\right\rangle_{\mathbf{k+l}} + \left\langle \dfrac{\partial^2 L}{\partial \dot{x}'\partial x}\right\rangle_{\mathbf{k-l}}\right)\right. \\ \qquad\qquad -\omega\cdot\mathbf{l}\left(\left\langle \dfrac{\partial^2 L}{\partial x'\partial \dot{x}}\right\rangle_{\mathbf{k+l}} + \left\langle \dfrac{\partial^2 L}{\partial x'\partial \dot{x}}\right\rangle_{\mathbf{k-l}}\right)\Big) \\ \dfrac{\partial^2 S_\omega}{\partial b'_{\mathbf{l}}\partial b_{\mathbf{k}}} = 2\mathcal{R}e\left(\left(-\left\langle \dfrac{\partial^2 L}{\partial x'\partial x}\right\rangle_{\mathbf{k+l}} + \left\langle \dfrac{\partial^2 L}{\partial x'\partial x}\right\rangle_{\mathbf{k-l}}\right)\right. \\ \qquad\qquad +(\omega\cdot\mathbf{k})(\omega\cdot\mathbf{l})\left(\left\langle \dfrac{\partial^2 L}{\partial \dot{x}'\partial \dot{x}}\right\rangle_{\mathbf{k+l}} + \left\langle \dfrac{\partial^2 L}{\partial \dot{x}'\partial \dot{x}}\right\rangle_{\mathbf{k-l}}\right)\Big) \\ \qquad +2\mathcal{I}m\left(\omega\cdot\mathbf{k}\left(-\left\langle \dfrac{\partial^2 L}{\partial \dot{x}'\partial x}\right\rangle_{\mathbf{k+l}} + \left\langle \dfrac{\partial^2 L}{\partial \dot{x}'\partial x}\right\rangle_{\mathbf{k-l}}\right)\right. \\ \qquad\qquad -\omega\cdot\mathbf{l}\left(\left\langle \dfrac{\partial^2 L}{\partial x'\partial \dot{x}}\right\rangle_{\mathbf{k+l}} + \left\langle \dfrac{\partial^2 L}{\partial x'\partial \dot{x}}\right\rangle_{\mathbf{k-l}}\right)\Big) \end{cases} \tag{A5.2}$$

and

$$\begin{cases} \dfrac{\partial^2 S_\omega}{\partial a'_{\mathbf{0}} \partial a_{\mathbf{k}}} = 2\mathcal{R}e \left\langle \dfrac{\partial^2 L}{\partial x' \partial x} \right\rangle_{\mathbf{k}} + 2\omega \cdot \mathbf{k}\, \mathcal{I}m \left\langle \dfrac{\partial^2 L}{\partial x' \partial \dot{x}} \right\rangle_{\mathbf{k}} \\ \dfrac{\partial^2 S_\omega}{\partial b'_{\mathbf{0}} \partial a_{\mathbf{k}}} = 0 \\ \dfrac{\partial^2 S_\omega}{\partial a'_{\mathbf{0}} \partial b_{\mathbf{k}}} = 2\mathcal{I}m \left\langle \dfrac{\partial^2 L}{\partial x' \partial x} \right\rangle_{\mathbf{k}} - 2\omega \cdot \mathbf{k}\mathcal{R}e \left\langle \dfrac{\partial^2 L}{\partial x' \partial \dot{x}} \right\rangle_{\mathbf{k}} \\ \dfrac{\partial^2 S_\omega}{\partial b'_{\mathbf{0}} \partial b_{\mathbf{k}}} = 0 \end{cases} \quad \text{and} \quad \begin{cases} \dfrac{\partial^2 S_\omega}{\partial a'_{\mathbf{0}} \partial a_{\mathbf{0}}} = \left\langle \dfrac{\partial^2 L}{\partial x' \partial x} \right\rangle_{\mathbf{0}} \\ \dfrac{\partial^2 S_\omega}{\partial a'_{\mathbf{0}} \partial b_{\mathbf{0}}} = 0 \\ \dfrac{\partial^2 S_\omega}{\partial b'_{\mathbf{0}} \partial b_{\mathbf{0}}} = 0 \end{cases}$$

One may notice that the terms in (A5.2) of the form $\langle \cdot \rangle_{\mathbf{k-l}}$ are not well defined since $\mathbf{k} - \mathbf{l}$ may not be in $\mathfrak{K}$. But this is not a problem since the derivatives of L are real so we can use

$$\mathcal{R}e\left(\langle \cdot \rangle_{\mathbf{k}}\right) + i\mathcal{I}m\left(\langle \cdot \rangle_{\mathbf{k}}\right) = \mathcal{R}e\left(\langle \cdot \rangle_{-\mathbf{k}}\right) - i\mathcal{I}m\left(\langle \cdot \rangle_{-\mathbf{k}}\right)$$

to express $\langle \cdot \rangle_{\mathbf{k-l}}$ using $\langle \cdot \rangle_{\mathbf{l-k}}$.

A5.2. Derivatives of the Lagrangian

We now express the derivatives of the Lagrangian

$$\begin{cases} L_1 = \dfrac{1}{2}\beta_1 \dot{r}_1^2 + \dfrac{\mu_1 \beta_1}{r_1} + \dfrac{1}{2}\beta_1 r_1^2 \xi^2 \\ L_2 = \dfrac{1}{2}\beta_2 \dot{r}_2^2 + \dfrac{\mu_2 \beta_2}{r_2} + \chi\left(\beta_1 r_1^2 \xi - \dfrac{1}{2}\beta_2 r_2^2 \chi\right) \\ L_{int} = -\dfrac{\mu_2 \beta_2}{r_2} + \dfrac{\mu_2 \beta_2}{\sqrt{\Delta_1}} + \dfrac{\mu}{\sqrt{\Delta_0}}. \end{cases}$$

We define

$$L_K := L_1 + L_2$$

which yields

$$\begin{cases} \dfrac{\partial L_K}{\partial r_1} = -\dfrac{\mu_1 \beta_1}{r_1^2} + \beta_1 r_1 \xi^2 \\ \dfrac{\partial L_K}{\partial r_2} = -\dfrac{\mu_2 \beta_2}{r_2^2} + \beta_2 r_2 \chi^2 \\ \dfrac{\partial L_K}{\partial \dot{w}} = \beta_1 r_1^2 \xi \end{cases} \quad \text{and} \quad \begin{cases} \dfrac{\partial L_K}{\partial \dot{r}_1} = \beta_1 \dot{r}_1 \\ \dfrac{\partial L_K}{\partial \dot{r}_2} = \beta_2 \dot{r}_2 \\ \dfrac{\partial L_K}{\partial w} = 0 \end{cases} \qquad \textbf{(A5.3)}$$

and

$$\begin{cases} \dfrac{\partial^2 L_K}{\partial r_1 \partial r_1} = \dfrac{2\mu_1\beta_1}{r_1^3} + \dfrac{\beta_1\left(\beta_2 r_2^2 - 3\beta_1 r_1^2\right)}{\beta_1 r_1^2 + \beta_2 r_2^2}\,\xi^2 \\ \dfrac{\partial^2 L_K}{\partial r_2 \partial r_1} = \dfrac{4\beta_1 r_1 \beta_2 r_2}{\beta_1 r_1^2 + \beta_2 r_2^2}\,\xi\chi \\ \dfrac{\partial^2 L_K}{\partial r_2 \partial r_2} = \dfrac{2\mu_2\beta_2}{r_2^3} + \dfrac{\beta_2\left(\beta_1 r_1^2 - 3\beta_2 r_2^2\right)}{\beta_1 r_1^2 + \beta_2 r_2^2}\,\chi^2 \end{cases} \tag{A5.4}$$

$$\begin{cases} \dfrac{\partial^2 L_K}{\partial \dot{w} \partial r_1} = \dfrac{2\beta_1 r_1 \beta_2 r_2^2}{\beta_1 r_1^2 + \beta_2 r_2^2}\,\xi \\ \dfrac{\partial^2 L_K}{\partial \dot{w} \partial r_2} = \dfrac{2\beta_1 r_1^2 \beta_2 r_2}{\beta_1 r_1^2 + \beta_2 r_2^2}\,\chi \end{cases} \quad \text{and} \quad \begin{cases} \dfrac{\partial^2 L_K}{\partial \dot{r}_1 \partial \dot{r}_1} = \beta_1 \\ \dfrac{\partial^2 L_K}{\partial \dot{r}_2 \partial \dot{r}_2} = \beta_2 \\ \dfrac{\partial^2 L_K}{\partial \dot{w} \partial \dot{w}} = \dfrac{\beta_1 r_1^2 \beta_2 r_2^2}{\beta_1 r_1^2 + \beta_2 r_2^2} \end{cases} \tag{A5.5}$$

The other derivatives of L_K are equal to 0. Furthermore

$$\begin{cases} \dfrac{\partial L_{int}}{\partial r_1} = -\dfrac{\mu_2\beta_2\delta_1\left(\delta_1 r_1 + r_2\cos w\right)}{\Delta_1(r_1, r_2, w)^{3/2}} - \dfrac{\mu\delta_0\left(\delta_0 r_1 + r_2\cos w\right)}{\Delta_0(r_1, r_2, w)^{3/2}} \\ \dfrac{\partial L_{int}}{\partial r_2} = \dfrac{\mu_2\beta_2}{r_2^2} - \dfrac{\mu_2\beta_2\left(r_2 + \delta_1 r_1\cos w\right)}{\Delta_1(r_1, r_2, w)^{3/2}} - \dfrac{\mu\left(r_2 + \delta_0 r_1\cos w\right)}{\Delta_0(r_1, r_2, w)^{3/2}} \\ \dfrac{\partial L_{int}}{\partial w} = \dfrac{\mu_2\beta_2\delta_1 r_1 r_2\sin w}{\Delta_1(r_1, r_2, w)^{3/2}} + \dfrac{\mu\delta_0 r_1 r_2\sin w}{\Delta_0(r_1, r_2, w)^{3/2}} \end{cases} \tag{A5.6}$$

and

$$
\begin{cases}
\dfrac{\partial^2 L_{int}}{\partial r_1 \partial r_1} = \dfrac{\mu_2 \beta_2 \delta_1^2}{\Delta_1(r_1,r_2,w)^{3/2}} \left(2 - \dfrac{3r_2^2 \sin^2(w)}{\Delta_1(r_1,r_2,w)}\right) + \dfrac{\mu \delta_0^2}{\Delta_0(r_1,r_2,w)^{3/2}} \left(2 - \dfrac{3r_2^2 \sin^2(w)}{\Delta_0(r_1,r_2,w)}\right) \\
\dfrac{\partial^2 L_{int}}{\partial r_2 \partial r_1} = \dfrac{\mu_2 \beta_2 \delta_1}{\Delta_1(r_1,r_2,w)^{3/2}} \left(2\cos w + \dfrac{3\delta_1 r_1 r_2 \sin^2(w)}{\Delta_1(r_1,r_2,w)}\right) \\
\qquad\qquad + \dfrac{\mu \delta_0}{\Delta_0(r_1,r_2,w)^{3/2}} \left(2\cos w + \dfrac{3\delta_0 r_1 r_2 \sin^2(w)}{\Delta_0(r_1,r_2,w)}\right) \\
\dfrac{\partial^2 L_{int}}{\partial w \partial r_1} = -\dfrac{\mu_2 \beta_2 \delta_1 r_2 \sin w}{\Delta_1(r_1,r_2,w)^{3/2}} \left(\dfrac{3\left(\delta_1 r_1 + r_2 \cos w\right)\delta_1 r_1}{\Delta_1(r_1,r_2,w)} - 1\right) \\
\qquad\qquad - \dfrac{\mu \delta_0 r_2 \sin w}{\Delta_0(r_1,r_2,w)^{3/2}} \left(\dfrac{3\left(\delta_0 r_1 + r_2 \cos w\right)\delta_0 r_1}{\Delta_0(r_1,r_2,w)} - 1\right) \\
\dfrac{\partial^2 L_{int}}{\partial r_2 \partial r_2} = -\dfrac{2\mu_2 \beta_2}{r_2^3} + \dfrac{\mu_2 \beta_2}{\Delta_1(r_1,r_2,w)^{3/2}} \left(2 - \dfrac{3\delta_1^2 r_1^2 \sin^2(w)}{\Delta_1(r_1,r_2,w)}\right) \\
\qquad\qquad + \dfrac{\mu}{\Delta_0(r_1,r_2,w)^{3/2}} \left(2 - \dfrac{3\delta_0^2 r_1^2 \sin^2(w)}{\Delta_0(r_1,r_2,w)}\right) \\
\dfrac{\partial^2 L_{int}}{\partial w \partial r_2} = -\dfrac{\mu_2 \beta_2 \delta_1 r_1 \sin w}{\Delta_1(r_1,r_2,w)^{3/2}} \left(\dfrac{3\left(r_2 + \delta_1 r_1 \cos w\right) r_2}{\Delta_1(r_1,r_2,w)} - 1\right) \\
\qquad\qquad - \dfrac{\mu \delta_0 r_1 \sin w}{\Delta_0(r_1,r_2,w)^{3/2}} \left(\dfrac{3\left(r_2 + \delta_0 r_1 \cos w\right) r_2}{\Delta_0(r_1,r_2,w)} - 1\right) \\
\dfrac{\partial^2 L_{int}}{\partial w \partial w} = \dfrac{\mu_2 \beta_2 \delta_1 r_1 r_2}{\Delta_1(r_1,r_2,w)^{3/2}} \left(\dfrac{3\delta_1 r_1 r_2 \sin^2(w)}{\Delta_1(r_1,r_2,w)} + \cos w\right) \\
\qquad\qquad + \dfrac{\mu \delta_0 r_1 r_2}{\Delta_0(r_1,r_2,w)^{3/2}} \left(\dfrac{3\delta_0 r_1 r_2 \sin^2(w)}{\Delta_0(r_1,r_2,w)} + \cos w\right)
\end{cases}
\tag{A5.7}
$$

The other derivatives of L_{int} are equal to 0.

To find these expressions we use the following equalities:

$$
\begin{cases}
\xi(r_1,r_2,\dot{w}) := \dfrac{\dot{w}\beta_2 r_2^2 + G}{\beta_1 r_1^2 + \beta_2 r_2^2} = \dot{v}_1 \\
\chi(r_1,r_2,\dot{w}) := \dfrac{\dot{w}\beta_1 r_1^2 - G}{\beta_1 r_1^2 + \beta_2 r_2^2} = \dot{v}_2
\end{cases}
$$

$$\begin{cases} \dfrac{\partial \chi}{\partial r_1} = \xi \dfrac{2\beta_1 r_1}{\beta_1 r_1^2 + \beta_2 r_2^2} \\ \dfrac{\partial \chi}{\partial r_2} = -\chi \dfrac{2\beta_2 r_2}{\beta_1 r_1^2 + \beta_2 r_2^2} \\ \dfrac{\partial \chi}{\partial \dot{w}} = \dfrac{\beta_1 r_1^2}{\beta_1 r_1^2 + \beta_2 r_2^2} \end{cases} \qquad \begin{cases} \dfrac{\partial \xi}{\partial r_1} = -\xi \dfrac{2\beta_1 r_1}{\beta_1 r_1^2 + \beta_2 r_2^2} = -\dfrac{\partial \chi}{\partial r_1} \\ \dfrac{\partial \xi}{\partial r_2} = \chi \dfrac{2\beta_2 r_2}{\beta_1 r_1^2 + \beta_2 r_2^2} = -\dfrac{\partial \chi}{\partial r_2} \\ \dfrac{\partial \xi}{\partial \dot{w}} = \dfrac{\beta_2 r_2^2}{\beta_1 r_1^2 + \beta_2 r_2^2} \end{cases}$$

$$\begin{cases} \dfrac{\partial^2 \chi}{\partial r_1 \partial r_1} = \xi \dfrac{2\beta_1}{\left(\beta_1 r_1^2 + \beta_2 r_2^2\right)^2} \left(\beta_2 r_2^2 - 3\beta_1 r_1^2\right) \\ \dfrac{\partial^2 \chi}{\partial r_2 \partial r_1} = -\left(\xi - \chi\right) \dfrac{4\beta_1 r_1 \beta_2 r_2}{\left(\beta_1 r_1^2 + \beta_2 r_2^2\right)^2} \\ \dfrac{\partial^2 \chi}{\partial \dot{w} \partial r_1} = \dfrac{2\beta_1 r_1 \beta_2 r_2^2}{\left(\beta_1 r_1^2 + \beta_2 r_2^2\right)^2} \\ \dfrac{\partial^2 \chi}{\partial r_2 \partial r_2} = \chi \dfrac{2\beta_2}{\left(\beta_1 r_1^2 + \beta_2 r_2^2\right)^2} \left(3\beta_2 r_2^2 - \beta_1 r_1^2\right) \\ \dfrac{\partial^2 \chi}{\partial \dot{w} \partial r_2} = -\dfrac{2\beta_1 r_1^2 \beta_2 r_2}{\left(\beta_1 r_1^2 + \beta_2 r_2^2\right)^2} \\ \dfrac{\partial^2 \chi}{\partial \dot{w} \partial \dot{w}} = 0 \end{cases}$$

$$\frac{\partial^2 \xi}{\partial x \partial x'} = -\frac{\partial^2 \chi}{\partial x \partial x'}$$

And

$$\begin{cases} \dfrac{\partial \Delta_i}{\partial r_1} = 2\delta_i(\delta_i r_1 + r_2 \cos w) \\ \dfrac{\partial \Delta_i}{\partial r_2} = 2(r_2 + \delta_i r_1 \cos w) \\ \dfrac{\partial \Delta_i}{\partial w} = -2\delta_i r_1 r_2 \sin w \end{cases} \qquad \begin{cases} \dfrac{\partial^2 \Delta_i}{\partial r_1 \partial r_1} = 2\delta_i^2 \\ \dfrac{\partial^2 \Delta_i}{\partial r_1 \partial r_2} = 2\delta_i \cos w \\ \dfrac{\partial^2 \Delta_i}{\partial r_1 \partial w} = -2\delta_i r_2 \sin w \\ \dfrac{\partial^2 \Delta_i}{\partial r_2 \partial r_2} = 2 \\ \dfrac{\partial^2 \Delta_i}{\partial r_2 \partial w} = -2\delta_i r_1 \sin w \\ \dfrac{\partial^2 \Delta_i}{\partial w \partial w} = -2\delta_i r_1 r_2 \cos w \end{cases}$$

A6. Results for the simple system

In this appendix we give the results of the extremalization algorithm using the quasi-Newton method applied to the system (5.3)

$$H(\phi_1, \phi_2, I_1, I_2) = \frac{I_1^2}{2} + \frac{I_2^2}{2} + \epsilon \left(\cos(\phi_1 + \phi_2) + \cos(\phi_1 - \phi_2)\right).$$

We take

$$\epsilon = 0.1,\ k_m = 16,\ \omega = (1, 0.615).$$

In the following table, we give the 20 largest harmonics produced by our algorithm given alongside the results of an independent numerical integration provided by J. Laskar. We give $\dot{\phi}_1 e^{i\phi_1}$

k_1	k_2	computed amplitude	compare
1	0	0.9402611215	0.9402611271
2	-1	0.3153598290	0.3153598289
0	1	0.1501049273	0.1501049272
3	-2	0.1045848880	0.1045848879
2	1	0.0470489075	0.0470489075
4	-3	0.0330532141	0.0330532141
3	0	0.0130709521	0.0130709521
0	-1	0.0110690954	0.0110690954
1	2	0.0104148751	0.0104148751
5	-4	0.0101128017	0.0101128017
-1	0	0.0046720595	0.0046720595
4	-1	0.0038272225	0.0038272225
6	-5	0.0030229781	0.0030229781
3	2	0.0014568229	0.0014568229
5	-2	0.0011064636	0.0011064636
1	-2	0.0010024737	0.0010024737
-1	2	0.0009160064	0.0009160064
7	-6	0.0008879908	0.0008879908
4	1	0.0003848985	0.0003848985
2	3	0.0003437432	0.0003437432

and $\dot{\phi}_2 e^{i\phi_2}$:

k_1	k_2	computed amplitude	compare
0	1	0.5782605897	0.5782605897
1	0	0.2440730526	0.2440730525
-1	2	0.0523702243	0.0523702242
1	2	0.0401220129	0.0401220129
-1	0	0.0179985290	0.0179985290
2	1	0.0122129590	0.0122129589
-2	3	0.0091585636	0.0091585637
-3	4	0.0082824759	0.0082824759
0	3	0.0080386356	0.0080386356
-2	1	0.0060366348	0.0060366348
2	-1	0.0055159514	0.0055159514
-4	5	0.0036828116	0.0036828116
0	-1	0.0028733166	0.0028733166
-3	2	0.0020019696	0.0020019696
-1	4	0.0016507370	0.0016507370
-5	6	0.0013538808	0.0013538808
2	3	0.0013242279	0.0013242279
-4	3	0.0006327064	0.0006327064
-6	7	0.0004547264	0.0004547264
3	2	0.0003781622	0.0003781622

The results are the same up to an error of 10^{-10}. The independent numerical integration gave frequencies equal to

$$\omega_{\text{num}} = (1.0000000001, 0.6149999999).$$

A7. Specifications of the computer used for computations

∗ CPU: 8×Intel® Xeon™ MP, clock: 3.66 GHz, cache: 1 MB per CPU, architecture: x86_64.
∗ Memory: 8 GB.
∗ Kernel: Linux 2.6.18-274.17.1.el5.

Programs

The programs provided in this appendix are written in the TRIP language, which is easily readable. The code listings below can therefore be seen as detailed descriptions of the algorithms we used.

We now give a rudimentary syntax guide for the TRIP language. Each line is terminated by either `$` or `;`. A subroutine (or *macro*) is defined by `macro name[args]{...};` and can be used by calling `%name[args]`. The symbol `//` comments the rest of the line. Comments can also be enclosed between `/*` and `*/`.

Numerical vectors (declared by `vnumR` in the real case and `vnumC` in the complex case), are vectors with numbers as entries. Tables (declared by `dim`) can contain any type of data, including series. Matrices are represented as tables of numerical vectors using the syntax `vnumR M[1:size];` followed by `resize(M,size);`.

The command `initcf` initializes an environment for Fourier series, in which the maximal order of the harmonics in given. The operations on Fourier series are subsequently truncated at that maximal order.

TRIP is a symbolic language, and any undeclared symbol in an expression is considered as a variable (as is the case for `M` in program P1).

P1. Kepler problem

The code for the program that computes the trajectories of a Kepler problem by extremalizing the action:

`keplactio.t`:

```
/*
Numerical computation of the trajectories of a Kepler problem
using a variational principle.
The parameters should be set in runExample and initIter.

The results are given in a vector rn: rn[j] holds the r computed
at the j-th step. RSA0 is the analytical solution (for comparison).
%norm gives the norm of a Fourier series, thus
  %norm[rn[j]-RsA0]
will give the "distance" of the solution from the analytical one.
To run, simply execute %runExample.

The algorithm first attempts to find a solution with a Lagrange
multiplier in its usual form to impose the initial condition. If
this fails, it then uses the modified Lagrange multiplier we
introduced.

I.Jauslin - last modified 26/04/2012
*/

//typical sequence of commands to run the algorithm
macro runExample{
//cutoff of the order of the harmonics
kMax=128$
%init[kMax];
//eccentricity
ee=0.4$
//maximal number of iterations using the usual Lagrange multiplier
Niter=20$
//maximal number of iterations using the modified Lagrange multiplier
NiterNew=300$
//stop the algorithm once it converged up to a precision of tol
tol=1e-12$
//stops the algorithm after maxCount convergent steps
maxCount=3$
//r to start the algorithm with
r0=1-ee*cos(M)$
//store it as a Fourier series
convcf(r0);
//Decompose it as a Fourier series
r0=%FourDecomp[r0]$
//run the program
```

```
43  %main[ee,kMax,r0,Niter,NiterNew];
44  };
45
46  //initializes the environment
47  macro init[kMax]{
48  //double precision floats
49  _modenum=NUMDBL$
50  //neglect anything below the given precision
51  _cleaneps=1e-25$
52  _cleanflag=1$
53
54  //size of a vector of Fourier coefficients
55  Nk=kMax+1$
56  //set the maximal order of Fourier series
57  initcf(X,-kMax,kMax)$
58  //define X as exp(iM)
59  X=expi(M,1,0)$
60  };
61
62  //runs the algorithm
63  macro main[ee,kMax,r0,nmax,nmaxNew]{
64  private rnp;
65  Niter=nmax$
66  %initIter[kMax,ee,r0]$
67  //runs the usual Lagrange multiplier algorithm
68  %iterOld[1,Niter]$
69  //if after trying with the usual Lagrange multiplier, DS!=0
70  if(sum(abs(%DAction[r]))>1e-10)then{
71  //modified Lagrange multiplier program
72    msg("newalg");
73  //extend the size of the vector rn
74    dim rnp[0:Niter+nmaxNew]$
75    rnp[0:Niter]=rn$
76    rn=rnp$
77  //if the old Lagrange multiplier failed to converge, reset
78  //to initial values
79    if(converges==0)then{r=r0$lam=0$};
80
81    %iterNew[Niter+1,Niter+nmaxNew]$
82    };
83  return(r)$
84  };
85
86  //sets constants and computes the analytical solution
87  macro initIter[kMax,ee,r0]{
88  //semi-major axis
89  a=1$
90  //mass of the star
91  mu=1$
```

```
92  //frequency
93  omega=sqrt(mu/a^3)$
94  //mass of the planet
95  B=1$
96  //square of the angular momentum
97  g2=B^2*mu*a*(1-ee^2)$
98
99  //analytical solution
100 RsA0=%rsa[kMax,ee,omega,mu,M]$
101 convcf(RsA0)$
102
103 //boundary condition
104 ri=a*(1-ee)$
105 //initialize r and lambda
106 r=r0$
107 lam=0$
108
109 //vector with r at each iteration step as a series
110 dim rn[0:Niter]$
111 rn[0]=%FourInvDecomp[r]$
112 convcf(rn[0]);
113 };
114
115 //analytical solution of the Kepler problem
116 macro rsa[kMax,ee,omega,mu,M]{
117 private a,E,cosEpM;
118 //semi-major axis
119 a=exp(1/3*log(mu/omega^2))$
120 //eccentric anomaly: initialize the algorithm
121 E=ee*sin(M)$
122 convcf(E)$
123
124 //algorithm to find the actual eccentric anomaly
125 for k=1 to 200{
126 //E_{n+1}=e*sin(E_n+M)
127 E=ee*real(sin(M)*cos(E)+cos(M)*sin(E))$
128 };
129 //r=a(1-e*cos(E+M))
130 cosEpM=real(cos(M)*cos(E)-sin(M)*sin(E))$
131 return(a*(1-ee*cosEpM))$
132 };
133
134
135 //the iteration with the usual Lagrange multiplier
136 macro iterOld[nmin,nmax]{
137 private j,stopCount,rlam,rdiff;
138 j=nmin$
139 //to stop the iteration after maxCount steps where r_n-r_{n+1}\approx0
140 stopCount=0$
```

```
141 while((j<=nmax)&&(stopCount<maxCount))do{
142   msg("step %2d\n",j);
143 //new r and lambda (Lagrange multiplier)
144   rlam=%iterFastLagrangeSym[r,lam]$
145   r=rlam[1:Nk]$
146   lam=rlam[Nk+1]$
147 //rn[j] is r as a series at the j-th step
148   rn[j]=%FourInvDecomp[r]$
149   convcf(rn[j]);
150
151 //the difference between the two latest steps
152   rdiff=%norm[rn[j]-rn[j-1]]$
153 //stop?
154   if(rdiff<tol)then{
155     stopCount=stopCount+1$
156     }
157     else{
158     stopCount=0$
159     };
160   j=j+1$
161   };
162 //if stopped, return to the last computed j
163 j=j-1$
164 //converged?
165 converges=0$
166 //if it converged before nmax steps, converges=1 and resize vectors
167 if(j<nmax)then{
168   rn=rn[0:j]$
169   Niter=j$
170   converges=1$
171   };
172 };
173
174 //the iteration with the modified Lagrange multiplier
175 macro iterNew[nmin,nmax]{
176 private j,stopCount,rlam,rdiff;
177 j=nmin$
178 //to stop the iteration after maxCount steps where r_n-r_{n+1}\approx0
179 stopCount=0$
180 while((j<=nmax)&&(stopCount<maxCount))do{
181   msg("step %2d\n",j);
182 //new r and lambda (Lagrange multiplier)
183   rlam=%iterFastLagrangeSymNew[r,lam]$
184   r=rlam[1:Nk]$
185   lam=rlam[Nk+1]$
186 //rn[j] is r as a series at the j-th step
187   rn[j]=%FourInvDecomp[r]$
188   convcf(rn[j]);
189
```

```
  rdiff=%norm[rn[j]-rn[j-1]]$
//stop?
  if(rdiff<tol)then{
    stopCount=stopCount+1$
    }
    else{
    stopCount=0$
    };
  j=j+1$
  };
j=j-1$
//converged?
//if it converged before nmax steps, resize vectors
if(j<nmax)then{
  rn=rn[0:j]$
  Niter=j$
  };
};

//iteration with the usual Lagrange multiplier
macro iterFastLagrangeSym[rr,llam]{
private rF,DS,D2S,Inv,rres,lres,ret,coefsDS,coefsD2S,k,l;
//Hessian of the action
//size Nk+1: the +1 is for the Lagrange multiplier
vnumR D2S[1:Nk+1]$
resize(D2S,Nk+1)$
//gradient of the action
vnumR DS$
resize(DS,Nk+1);
//r as a series
rF=%FourInvDecomp[rr]$
convcf(rF);
//Fourier coefficients of the gradient and Hessian of the Lagrangian
coefsDS=%FourCoefs[(g2-B^2*mu*rF)/(B*rF^3)]$
coefsD2S=%FourCoefs[(-3*g2+2*B^2*mu*rF)/(B*rF^4)]$
//make D2S and DS
DS[1]=coefsDS[0]+llam$
D2S[1][1]=coefsD2S[0]$
D2S[1][Nk+1]=1$
D2S[Nk+1][1]=1$
for kn=2 to Nk{
//kn is the index in the vector representation of a Fourier series
//k is the order of the harmonic kn corresponds to
  k=kn-1$
  DS[kn]=2*omega^2*k^2*B*rr[kn]+2*coefsDS[k]+2*llam$
  D2S[kn][1]=2*coefsD2S[k]$
  D2S[1][kn]=D2S[kn][1]$
//only take the 2*k term if it is not neglected
  if(2*k<=kMax)then{
```

```
239       D2S[kn][kn]=2*omega^2*k^2*B+2*coefsD2S[0]+2*coefsD2S[2*k]$
240       }
241     else{
242   //no 2*k term
243       D2S[kn][kn]=2*omega^2*k^2*B+2*coefsD2S[0]$
244       };
245   //we only fill the upper triangular part of the (symmetric) Hessian
246     for ln=kn+1 to Nk{
247       l=ln-1$
248       if(k+l<=kMax)then{
249         D2S[kn][ln]=2*coefsD2S[k+l]+2*coefsD2S[k-l]$
250         }
251       else {
252         D2S[kn][ln]=2*coefsD2S[k-l]$
253         };
254   //symmetrize
255       D2S[ln][kn]=D2S[kn][ln]$
256       };
257     D2S[kn][Nk+1]=2$
258     D2S[Nk+1][kn]=2$
259     };
260   D2S[Nk+1][Nk+1]=0$
261   DS[Nk+1]=%r0[rr]-ri$
262   //inverse
263   Inv=D2S^-1$
264   y=%matProd[Inv,DS]$
265
266   rres=rr-y[1:Nk]$
267   lres=llam-y[Nk+1]$
268   ret=vnumR[rres:lres]$
269   return(ret)$
270   };
271
272   //iteration with the modified Lagrange multiplier
273   macro iterFastLagrangeSymNew[rr,llam]{
274   private rF,DS,D2S,Inv,rres,lres,ret,coefsDS,coefsD2S,k,l,rz;
275   //Hessian of the action
276   //size Nk+1: the +1 is for the Lagrange multiplier
277   vnumR D2S[1:Nk+1]$
278   resize(D2S,Nk+1)$
279   //gradient of the action
280   vnumR DS$
281   resize(DS,Nk+1);
282   //r as a series
283   rF=%FourInvDecomp[rr]$
284   convcf(rF);
285   //Fourier coefficients of the gradient and Hessian of the Lagrangian
286   coefsDS=%FourCoefs[(g2-B^2*mu*rF)/(B*rF^3)]$
287   coefsD2S=%FourCoefs[(-3*g2+2*B^2*mu*rF)/(B*rF^4)]$
```

```
288 //make D2S and DS
289 //r(M=0)-r_{initial}
290 rz=%r0[rr]-ri$
291 //to bypass the fact that D2S is not invertible when rz=0
292 if(rz!=0)then{
293   DS[1]=coefsDS[0]+llam*rz$
294   D2S[1][1]=coefsD2S[0]+llam$
295   D2S[1][Nk+1]=rz$
296   D2S[Nk+1][1]=rz$
297   for kn=2 to Nk{
298 //kn is the index in the vector representation of a Fourier series
299 //k is the order of the harmonic kn corresponds to
300     k=kn-1$
301     DS[kn]=2*omega^2*k^2*B*rr[kn]+2*coefsDS[k]+2*llam*rz$
302     D2S[kn][1]=2*coefsD2S[k]+2*llam$
303     D2S[1][kn]=D2S[kn][1]$
304 //only take the 2*k term if it is not neglected
305     if(2*k<=kMax)then{
306       D2S[kn][kn]=2*omega^2*k^2*B+2*coefsD2S[0]+2*coefsD2S[2*k]+4*llam$
307       }
308     else{
309 //no 2*k term
310       D2S[kn][kn]=2*omega^2*k^2*B+2*coefsD2S[0]+4*llam$
311       };
312 //we only fill the upper triangular part of the (symmetric) Hessian
313     for ln=kn+1 to Nk{
314       l=ln-1$
315       if(k+l<=kMax)then{
316         D2S[kn][ln]=2*coefsD2S[k+l]+2*coefsD2S[k-l]+4*llam$
317         }
318       else {
319         D2S[kn][ln]=2*coefsD2S[k-l]+4*llam$
320         };
321 //symmetrize
322       D2S[ln][kn]=D2S[kn][ln]$
323       };
324     D2S[kn][Nk+1]=2*rz$
325     D2S[Nk+1][kn]=2*rz$
326     };
327   D2S[Nk+1][Nk+1]=llam$
328   DS[Nk+1]=rz^2/2+llam^2/2$
329 //inverse
330   Inv=D2S^-1$
331   y=%matProd[Inv,DS]$
332
333   rres=rr-y[1:Nk]$
334   lres=llam-y[Nk+1]$
335   }
336 else{
```

```
337 //in this case D2S is only invertible if the Nk+1-th term is neglected
338   msg("rz=0");
339 //resize DS and D2S
340   vnumR D2S[1:Nk]$
341   resize(D2S,Nk);
342   resize(DS,Nk);
343
344   DS[1]=coefsDS[0]+llam*rz$
345   D2S[1][1]=coefsD2S[0]+llam$
346   for kn=2 to Nk{
347     k=kn-1$
348     DS[kn]=2*omega^2*k^2*B*rr[kn]+2*coefsDS[k]+2*llam*rz$
349     D2S[kn][1]=2*coefsD2S[k]+2*llam$
350     D2S[1][kn]=D2S[kn][1]$
351     if(2*k<=kMax)then{
352       D2S[kn][kn]=2*omega^2*k^2*B+2*coefsD2S[0]+2*coefsD2S[2*k]+4*llam$
353       }
354     else{
355       D2S[kn][kn]=2*omega^2*k^2*B+2*coefsD2S[0]+4*llam$
356       };
357     for ln=kn+1 to Nk{
358       l=ln-1$
359       if(k+l<=kMax)then{
360         D2S[kn][ln]=2*coefsD2S[k+l]+2*coefsD2S[k-l]+4*llam$
361         }
362       else {
363         D2S[kn][ln]=2*coefsD2S[k-l]+4*llam$
364         };
365       D2S[ln][kn]=D2S[kn][ln]$
366       };
367     };
368   Inv=D2S^-1$
369   y=%matProd[Inv,DS]$
370
371   rres=rr-y[1:Nk]$
372 //update lam=0
373   lres=0$
374   };
375 ret=vnumR[rres:lres]$
376 return(ret)$
377 };
378
379 //returns the initial value for a given r
380 macro r0[rr]{
381 return(rr[1]+2*sum(rr[2:Nk]))$
382 };
383
384 //matrix product of a square matrix with a column vector
385 macro matProd[M,x]{
```

```
386 private ret;
387 vnumR ret$
388 resize(ret,size(M))$
389 for n=1 to size(M){
390   ret[n]=real(sum(M[n]*x))$
391   };
392 return(ret)$
393 };
394
395 //Fourier series decomposition
396 //only the k>=0 are considered
397 macro FourDecomp[f]{
398 private ff;
399 vnumR ff$
400 resize(ff,Nk)$
401 cfcoef_tabexp(f,fcoefs,ffexp)$
402 //fcoefs is a vector with the coefficients corresponding to the
403 //exponents in ffexp[1] (which is also a vector).
404 for l=1 to size(fcoefs) {
405   if(ffexp[1][l]>=0)then{
406     ff[ffexp[1][l]+1]=real(fcoefs[l])$
407     };
408   };
409 return(ff)$
410 };
411 //Inverse
412 macro FourInvDecomp[ff]{
413 private f;
414 f=ff[1]$
415 for l=2 to Nk {
416   f=f+ff[l]*(X^(l-1)+X^(1-l))$
417   };
418 return(f)$
419 };
420 //Fourier coefficients
421 //with all the terms (e.g. ff[-3] is the -3 harmonic)
422 macro FourCoefs[f]{
423 private ff;
424 dim ff[-kMax:kMax]$
425 cfcoef_tabexp(f,fcoefs,ffexp)$
426 for l=1 to size(fcoefs) {
427   ff[ffexp[1][l]]=real(fcoefs[l])$
428   };
429 return(ff)$
430 };
431
432 //gradient of the action as a vector
433 macro DAction [rr]{
434 private DS,rF,k;
```

```
435 vnumR DS$
436 resize(DS,Nk);
437 rF=%FourInvDecomp[rr]$
438 convcf(rF)$
439 coefs=%FourCoefs[(g2-B^2*mu*rF)/(B*rF^3)]$
440 DS[1]=coefs[0]$
441 for kn=2 to Nk{
442   k=kn-1$
443   DS[kn]=2*omega^2*B*k^2*rr[kn]+2*coefs[k]$
444   };
445 return(DS)$
446 };
447
448 //norm of a Fourier transformable function: \sum |c_k|
449 macro norm[u]{
450 private nor;
451 cfcoef_tabexp(u,ucoef,uexp)$
452 nor=sum(abs(ucoef))$
453 return(nor)$
454 };
```

P2. Van Wijngaarden-Dekker-Brent algorithm

The code for the implementation of the Van Wijngaarden-Dekker-Brent algorithm to compute a zero of a function:

brentZero:

```
/*
An implementation of the Van Wijngaarden-Dekker-Brent method
for finding a zero of a function. The function must be defined
a priori in a macro called func that takes one real argument and returns
a real value.

The algorithm first brackets the zero, in the sense that it
finds an interval in which the zero is, and then searches
for it inside the interval.

I.Jauslin - last modified 26/04/2012
*/


//x1 and x2 are the starting points, tol is the tolerance: the
// algorithm stops if |f(b)|<tol, and d is a multiplication factor
//used in the initial bracketing: the algorithms searches for
//the bracket by multiplying its length by d, initfunca is the
//initial value of the function func at point x1;
//brackmax is the maximal number of iterations to find the
//initial bracket; itmax is the maximal number of iterations
//to find the zero.
//The macro requires to have a macro func of one variable
//previously defined
macro brentZero[x1,x2,tol,d,initfunca,brackmax,itmax]{
private a,b,c,R,S,T,P,Q,fa,fb,fc,L,maxiter,n,delta;
//the previous approximation
a=x1$
//the approximation
b=x2$
//extra point
c=a$
fa=initfunca$
fb=%func[b]$
fc=fa$

//initial bracketting
n=1$
//maximum number of attempts to bracket
maxiter=brackmax$
while((fa*fb>0)&&(n<maxiter))do{
  if(abs(fa)<abs(fb))then{
```

```
43  //expand to the left
44      a=b+(a-b)*d$
45      fa=%func[a]$
46      }
47    else{
48  //expand to the right
49      b=a+(b-a)*d$
50      fb=%func[b]$
51      };
52    n=n+1$
53    };
54  n=1$
55  //if the zero still isn't bracketed, try contracting
56  if(fa*fb>0)then{
57    b=x2$
58    while((fa*fb>0)&&(n<maxiter))do{
59      if(abs(fa)<abs(fb))then{
60  //contract from the left
61        a=b+(a-b)/d$
62        fa=%func[a]$
63        }
64      else{
65  //contract from the right
66        b=a+(b-a)/d$
67        fb=%func[b]$
68        };
69      n=n+1$
70      };
71    };
72  n=1$
73  //return a full step if the zero can't be bracketed
74  if(fa*fb>0)then{
75    msg("error: can't bracket zero, f(1)=%g\n",%func[1]);
76    return(x2);
77    exit;
78  };
79  reta=a$
80  retb=b$
81
82  //size of the bracket
83  L=b-a$
84  //increment
85  delta=L$
86  //maximum number of iterations
87  maxiter=itmax$
88  //loop while |f(b)|>tol
89  while((abs(fb)>tol)&&(n<maxiter))do{
90  //get c and a on the same side
91    if(fb*fc>0)then{
```

```
92      c=a$
93      fc=fa$
94      L=b-a$
95      delta=L$
96      };
97    //b should be closer to the zero than c
98    if(abs(fc)<abs(fb))then{
99      a=b$
100     b=c$
101     c=a$
102     fa=fb$
103     fb=fc$
104     fc=fa$
105     };
106   S=fb/fa$
107   if(a==c)then{
108     P=(c-b)*S$
109     Q=1-S$
110     }
111   else{
112     T=fa/fc$
113     R=fb/fc$
114     P=S*((c-b)*T*(T-R)-(b-a)*(R-1))$
115     Q=(T-1)*(R-1)*(S-1)$
116     };
117   //conditions to keep b within the brackets
118   if(P>0)then{Q=-Q$};
119   P=abs(P)$
120   //conditions to accept the step
121   if((2*P<3/2*(c-b)*Q)&&(2*P<abs(L*Q)))then{
122     L=delta$
123     delta=P/Q$
124     }
125   else{
126   //if step failed, bissection method
127     delta=(c-b)/2$
128     L=delta$
129     };
130   //update new points
131   a=b$
132   fa=fb$
133   b=b+delta$
134   fb=%func[b]$
135
136   n=n+1$
137   };
138 if(n==maxiter)then{
139 //failed, use the last attempted b
140   msg("max iterations, f(%g)=%g with a=%g and b=%g\n",b,%func[b],reta,retb);
```

```
141     }
142   else{
143   //success
144     msg("found f(%g)=%g after %d tries with a=%g and b=%g\n",b,fb,n,reta,retb);
145     };
146   return(b)$
147   };
```

P3. Three body problem

The code for the program that computes invariant tori for the three body problem using the quasi-Newton algorithm:

`threebodies_quasiNewt.t`:

```
/*
Numerical computation of invariant tori for the Sun-Jupiter-Saturn
system using a variational principle.
The parameters should be set in runExample and initIter.
The model is defined by the derivatives of the Lagrangian specified
in the macros of the form DS* and D2S*.
Uses a Quasi-Newton method.

Requires two auxiliary files: brentZero.t that computes the zero of a real
function of one variable; and inverseBlock6.t that inverts a matrix
made of 36 diagonal blocks.

To run, execute %runExample.
The results are given in the vectors rn_1, rn_2 and wn, containing the
positions as series computed at each step. DSn is a vector containing
the norm of the gradient of the action at each step.

I.Jauslin - last modified 26/04/2012
*/

//include brentZero.t
//include inverseBlock6.t

//typical sequence of commands to run the algorithm
macro runExample{
//cutoff of the order of the harmonics
kMax=8$
%init[kMax];
//maximal number of iterations
Niter=2$
//stop the algorithm once it converged up to a precision of tol
tolerance=1e-18$
//stops the algorithm after maxCount convergent steps
maxCount=3$
//run the program
%main[kMax,Niter];
};


//initialize the environment
macro init[kMax]{
//double precision floats
```

```
43 _modenum=NUMDBL$
44 //neglect anything below the given precision
45 _cleaneps=1e-25$
46 _cleanflag=1$
47
48 //size of a vector of Fourier coefficients
49 Nku=kMax*(2*kMax+1)^2+kMax*(2*kMax+1)+(kMax+1)$
50
51 //declare the three M's as variables
52 dimvar M[1:3]$
53 tabvar(M)$
54 //declare X_j=exp(iM_j)
55 dimvar X[1:3]$
56 tabvar(X)$
57 for j=1 to size(X){
58   X[j]=expi(M[j],1,0)$
59   };
60 //set the maximal order of Fourier series
61 initcf((X_1,-kMax,kMax),(X_2,-kMax,kMax),(X_3,-kMax,kMax))$
62 };
63
64 //runs the algorithm
65 macro main[kMax,Niter]{
66 %initIter[kMax]$
67 %iter[1,Niter]$
68 };
69
70 //sets constants
71 macro initIter[kMax]{
72 //masses are expressed in multiples of Saturn's mass
73 //durations in years
74 //lengths in Astronomical Units (semi-major axis of the Earth)
75 //angles in radiants
76
77 //the three frequencies
78 omega=vnumR[0.529695:0.213265:-0.000114735]$
79 //the vector that defines the linear combination of v_1 and v_2 in w:
80 //w=v_1-v_2
81 W=vnumR[1:-1:0]$
82
83 //masses
84 m_0=3.49789799959e3$$
85 m_1=3.33976481134$
86 m_2=1$
87 //gravitational constant
88 Gr=2.959122082855911E-004*365.25^2*2.858859807E-004$
89
90 //constants
91 B_1=m_0*m_1/(m_0+m_1)$
```

```
92  B_2=m_2*(m_0+m_1)/(m_0+m_1+m_2)$
93  mu_1=Gr*(m_0+m_1)$
94  mu_2=Gr*m_0*(m_0+m_1+m_2)/(m_0+m_1)$
95  mu=Gr*m_1*m_2$
96  delta_1=m_1/(m_0+m_1)$
97  delta_0=m_0/(m_0+m_1)$
98
99  //semi-major axes
100 a_1=exp(1/3*log(mu_1/omega[1]^2))$
101 a_2=exp(1/3*log(mu_2/omega[2]^2))$
102 //eccentricities (only used to fix the total angular momentum)
103 e_1=0.04814707261917873$
104 e_2=0.05381979488308911$
105 //total angular momentum
106 G=B_1*sqrt(mu_1*a_1*(1-e_1^2))+B_2*sqrt(mu_2*a_2*(1-e_2^2))$
107
108 //analytical solution of a non-interacting torus
109 //stores the analytical r_1 in rsa_1, r_2 in rsa_2,
110 //v_1 in vsa_1, v_2 in vsa_2
111 msg("rsa");
112 %getRsa;
113 //to start the algorithm with
114 //added a term in cos(M_3) since the iteration would
115 //not create such a term
116 r01=%SeriesToVect[rsa_1-0.0001*cos(M_3)]$
117 r02=%SeriesToVect[rsa_2-0.0001*cos(M_3)]$
118 //SeriesToVectW must be used for the angles
119 // it gets rid of the linear part (omega_1*M_1-omega_2*M_2)
120 v01=%SeriesToVectW[vsa_1]$
121 v02=%SeriesToVectW[vsa_2]$
122 w0=v01-v02$
123
124 r_1=r01$
125 r_2=r02$
126 w=w0$
127 //initialize H
128 msg("Hinit");
129 %Hinit$
130
131 //vectors in which we store the results and the gradient of the action
132 dim rn_1[0:Niter]$
133 dim rn_2[0:Niter]$
134 dim wn[0:Niter]$
135 vnumR DSn$resize(DSn,Niter);
136
137 //initialize the vectors
138 rn_1[0]=%VectToSeries[r_1]$
139 convcf(rn_1[0]);
140 rn_2[0]=%VectToSeries[r_2]$
```

```
141 convcf(rn_2[0]);
142 wn[0]=%VectToSeriesW[w]$
143 convcf(wn[0]);
144 };
145
146 //get the analytical solution of the non-interacting problem
147 macro getRsa{
148 rsa_1=%r[kMax,e_1,omega[1],mu_1,M_1]$
149 vsa_1=integ(%dv[rsa_1,e_1,a_1],M_1)$
150 rsa_2=%r[kMax,e_2,omega[2],mu_2,M_2]$
151 vsa_2=integ(%dv[rsa_2,e_2,a_2],M_2)$
152 };
153
154 //the analytical r
155 macro r[kMax,ee,omega,mu,M]{
156 private a,E,cosEpM;
157 //semi-major axis
158 a=exp(1/3*log(mu/omega^2))$
159 //eccentric anomaly: initialize the algorithm
160 E=ee*sin(M)$
161 convcf(E)$
162 //algorithm to find the actual eccentric anomaly
163 for k=1 to 200{
164 //E_{n+1}=e*sin(E_n+M)
165 E=ee*real(sin(M)*cos(E)+cos(M)*sin(E))$
166 };
167 //r=a(1-e*cos(E+M))
168 cosEpM=real(cos(M)*cos(E)-sin(M)*sin(E))$
169 return(a*(1-ee*cosEpM))$
170
171 };
172
173 //the analytical \dot v
174 macro dv[r,ee,a]{
175 return(a^2*sqrt(1-ee^2)/r^2)$
176 };
177
178
179 //initialize H
180 macro Hinit{
181 private rF_1,rF_2,wF,D2Sr1r1,D2Sr1r2,D2Sr1w,D2Sr2r2,D2Sr2w,D2Sww;
182 //declare H
183 //the size is 6*Nku since there is 2 Nku per variable (1 for the
184 //real and 1 for the imaginary part)
185 vnumR H[1:6*Nku]$
186 resize(H,6*Nku,0)$
187 //r's and w as series
188 rF_1=%VectToSeries[r_1]$
189 convcf(rF_1);
```

```
rF_2=%VectToSeries[r_2]$
convcf(rF_2);
//VectToSeriesW adds the linear part (omega_1*M_1-omega_2*M_2)
wF=%VectToSeriesW[w]$
convcf(wF)$

//the blocks in the Hessian of the action
D2Sr1r1=%D2Sr1r1[rF_1,rF_2,wF]$
D2Sr1r2=%D2Sr1r2[rF_1,rF_2,wF]$
D2Sr1w=%D2Sr1w[rF_1,rF_2,wF]$
D2Sr2r2=%D2Sr2r2[rF_1,rF_2,wF]$
D2Sr2w=%D2Sr2w[rF_1,rF_2,wF]$
D2Sww=%D2Sww[rF_1,rF_2,wF]$
//the Hessian of the action
//it is a degree three tensor: D2S[i,j][k] is the element
//of the block (i,j) on the diagonal position (k,k)
vnumR D2S[1:6,1:6]$
//put the blocks in D2S
D2S[1:2,1:2]=D2Sr1r1$
D2S[1:2,3:4]=D2Sr1r2$
D2S[1:2,5:6]=D2Sr1w$
D2S[3:4,1:2]=%transpose[D2Sr1r2]$
D2S[3:4,3:4]=D2Sr2r2$
D2S[3:4,5:6]=D2Sr2w$
D2S[5:6,1:2]=%transpose[D2Sr1w]$
D2S[5:6,3:4]=%transpose[D2Sr2w]$
D2S[5:6,5:6]=D2Sww$

//must correct the term in b_0: all the b_0=0, so we put ones on the
//diagonal terms corresponding to b_0 and 0 on the non-diagonal terms
//thus D2S is invertible, but does not touch the b_0 term
for n=1 to 3{
  D2S[2*n,2*n][1]=1$
  };

//inverse using the formula in inverseBlock6.t
Inv=%inverseBlock[D2S]$

//set H to the inverse
for n=0 to 5{
  for m=0 to 5{
    for k=1 to Nku{
      H[n*Nku+k][m*Nku+k]=Inv[n+1,m+1][k]$
      };
    };
  };
};

//the transpose of a degree three tensor with respect to its
```

```
239 //two first components: M[i,j][k]^T=M[j,i][k]
240 macro transpose[M]{
241 private ret;
242 ret=M$
243 ret[1,2]=M[2,1]$
244 ret[2,1]=M[1,2]$
245 return(ret)$
246 };
247
248
249 //The iteration
250 macro iter[nmin,nmax]{
251 private j,stopCount,rdiff_1,rdiff_2,wdiff;
252 j=nmin$
253 stopCount=0$
254 while((j<=nmax)&&(stopCount<maxCount))do{
255   msg("step %2d\n",j);
256 //run the algorithm: updates r_1,r_2,w and H
257   %iterThreeBodies$
258 //fill the results vectors
259   rn_1[j]=%VectToSeries[r_1]$
260   convcf(rn_1[j]);
261   rn_2[j]=%VectToSeries[r_2]$
262   convcf(rn_2[j]);
263   wn[j]=%VectToSeriesW[w]$
264   convcf(wn[j]);
265 //the norm of the gradient of the action at the latest step
266   DSn[j]=sum(abs(DS))$
267
268 //the difference between the two latest steps
269   rdiff_1=%norm[rn_1[j]-rn_1[j-1]]$
270   rdiff_2=%norm[rn_2[j]-rn_2[j-1]]$
271   wdiff=%norm[wn[j]-wn[j-1]]$
272
273 //stop?
274   if((rdiff_1<tolerance)&&(rdiff_2<tolerance)&&//
275       (wdiff<tolerance))then
276     {stopCount=stopCount+1$}else{stopCount=0$};
277   j=j+1$
278   };
279 };
280
281 //The algorithm: changes r_1,r_2,w and H from their value to the next
282 macro iterThreeBodies{
283 private lam,rnew,xn,sn,xs,Hs,sHs;
284 //the gradient of the action before the step is performed
285 DS=%DAction[r_1,r_2,w]$
286 //the three variables concatenated
287 ro=vnumR[r_1:r_2:w]$
```

```
288 //H*DS, is not private since it must be called by func
289 //the brackets around H make it so it is not copied in RAM
290 hds=%matProd[[H],DS]$
291 //lamb that extremalizes the action in the direction hds
292 //calls func
293 //macro in brentZero.t
294 //0,0.1 are the first guesses to bracket the extremum
295 //1e-12 is the required precision, 2 is a parameter
296 //5 is the maximum number of attempts to bracket the extremum
297 //20 is the maximum number of steps to compute it
298 lam=%brentZero[0,0.1,1e-12,2,%scalar[hds,DS],5,20]$
299 //new r_1,r_2 and w
300 rnew=ro-lam*hds$
301
302 //x_n
303 xn=rnew-ro$
304 //s_n, DSnew is set by evaluating func in %brentZero
305 sn=DSnew-DS$
306 xs=%scalar[xn,sn]$
307 Hs=%matProd[[H],sn]$
308 sHs=%scalar[sn,Hs]$
309 //update H
310 for k=1 to 6*Nku{
311   H[k]=H[k]+xn[k]*xn/xs*(1+sHs/xs)-(Hs[k]*xn+xn[k]*Hs)/xs$
312   };
313
314 //update r_1,r_2 and w
315 r_1=rnew[:2*Nku]$
316 r_2=rnew[2*Nku+1:4*Nku]$
317 w=rnew[4*Nku+1:6*Nku]$
318 };
319
320 //the function to be extremalized by %brentZero
321 //(H\partial S)*\partial S(r_n-lam*H\partial S)
322 //the gradient of the action in the proposed new r_1,r_2 and w
323 //is stored in DSnew which is then used by %iterThreeBodies
324 macro func[lam]{
325 DSnew=%DAction[ro[1:2*Nku]-lam*hds[1:2*Nku]//
326     ,ro[2*Nku+1:4*Nku]-lam*hds[2*Nku+1:4*Nku]//
327     ,ro[4*Nku+1:6*Nku]-lam*hds[4*Nku+1:6*Nku]]$
328 return(%scalar[hds,DSnew])$
329 };
330
331
332 //gradient of the action
333 macro DAction[r_1,r_2,w]{
334 private rF_1,rF_2,wF,DSr1,DSr2,DSw,DS;
335 //r_1,r_2 and w as Fourier series
336 rF_1=%VectToSeries[r_1]$
```

```
337 convcf(rF_1);
338 rF_2=%VectToSeries[r_2]$
339 convcf(rF_2);
340 wF=%VectToSeriesW[w]$
341 convcf(wF)$
342
343 //gradient
344 DSr1=%DSr1[rF_1,rF_2,wF]$
345 DSr2=%DSr2[rF_1,rF_2,wF]$
346 DSw=%DSw[rF_1,rF_2,wF]$
347 DS=vnumR[DSr1:DSr2:DSw]$
348 return(DS)$
349 };
350
351
352
353 //The derivatives of the Lagrangian
354
355 macro DSr1[r_1,r_2,w]{
356 private fk,fi1,fi0,DS,xi;
357 xi=%xi[r_1,r_2,%Dw[w]]$
358 fk=-mu_1*B_1/r_1^2+B_1*r_1*xi^2$
359 fi1=-mu_2*B_2*delta_1*(delta_1*r_1+r_2*%cosw[w])/sqrt(%Delta_1[r_1,r_2,w])^3$
360 fi0=-mu*delta_0*(delta_0*r_1+r_2*%cosw[w])/sqrt(%Delta_0[r_1,r_2,w])^3$
361 DS=%DS[fk+fi1+fi0,B_1*%Dw[r_1]]$
362 return(DS)$
363 };
364
365 macro DSr2[r_1,r_2,w]{
366 private fk,fi1,fi0,DS,chi;
367 chi=%chi[r_1,r_2,%Dw[w]]$
368 fk=-mu_2*B_2/r_2^2+B_2*r_2*chi^2$
369 fi1=mu_2*B_2/r_2^2-mu_2*B_2*(r_2+delta_1*r_1*%cosw[w])/sqrt(%Delta_1[r_1,r_2,w])^3$
370 fi0=-mu*(r_2+delta_0*r_1*%cosw[w])/sqrt(%Delta_0[r_1,r_2,w])^3$
371 DS=%DS[fk+fi1+fi0,B_2*%Dw[r_2]]$
372 return(DS)$
373 };
374
375 macro DSw[r_1,r_2,w]{
376 private fkdx,fi1,fi0,DS,xi;
377 xi=%xi[r_1,r_2,%Dw[w]]$
378 fkdx=B_1*r_1^2*xi$
379 fi1=-mu_2*B_2*delta_1*r_1*r_2*%sinw[w]/sqrt(%Delta_1[r_1,r_2,w])^3$
380 fi0=-mu*delta_0*r_1*r_2*%sinw[w]/sqrt(%Delta_0[r_1,r_2,w])^3$
381 DS=%DS[fi1+fi0,fkdx]$
382 return(DS)$
383 };
384
385
```

```
386 macro D2Sr1r1[r_1,r_2,w]{
387 private D2S,fk,fi1,fi0,fdxdx,xi;
388 xi=%xi[r_1,r_2,%Dw[w]]$
389 fk=2*mu_1*B_1/r_1^3+B_1*xi^2//
390     *(B_2*r_2^2-3*B_1*r_1^2)/(B_1*r_1^2+B_2*r_2^2)$
391 fi1=mu_2*B_2*delta_1^2/sqrt(%Delta_1[r_1,r_2,w])^3//
392     *(2-3*r_2^2*%sinw[w]^2/%Delta_1[r_1,r_2,w])$
393 fi0=mu*delta_0^2/sqrt(%Delta_0[r_1,r_2,w])^3//
394     *(2-3*r_2^2*%sinw[w]^2/%Delta_0[r_1,r_2,w])$
395 fdxdx=B_1$convcf(fdxdx);
396 D2S=%D2S[fk+fi1+fi0,0,0,fdxdx]$
397 return(D2S)$
398 };
399
400 macro D2Sr1r2[r_1,r_2,w]{
401 private D2S,fk,fi1,fi0,xi,chi;
402 xi=%xi[r_1,r_2,%Dw[w]]$
403 chi=%chi[r_1,r_2,%Dw[w]]$
404 fk=4*B_1*r_1*B_2*r_2/(B_1*r_1^2+B_2*r_2^2)*xi*chi$
405 fi1=mu_2*B_2*delta_1/sqrt(%Delta_1[r_1,r_2,w])^3//
406     *(2*%cosw[w]+3*delta_1*r_1*r_2*%sinw[w]^2/%Delta_1[r_1,r_2,w])$
407 fi0=mu*delta_0/sqrt(%Delta_0[r_1,r_2,w])^3//
408     *(2*%cosw[w]+3*delta_0*r_1*r_2*%sinw[w]^2/%Delta_0[r_1,r_2,w])$
409 D2S=%D2S[fk+fi1+fi0,0,0,0]$
410 return(D2S);
411 };
412
413 macro D2Sr1w[r_1,r_2,w]{
414 private D2S,fkdxx,fi1,fi0,xi;
415 xi=%xi[r_1,r_2,%Dw[w]]$
416 fkdxx=2*B_1*r_1*B_2*r_2^2/(B_1*r_1^2+B_2*r_2^2)*xi$
417 fi1=-mu_2*B_2*delta_1*r_2*%sinw[w]/sqrt(%Delta_1[r_1,r_2,w])^3//
418     *(3*(delta_1*r_1+r_2*%cosw[w])*delta_1*r_1/%Delta_1[r_1,r_2,w]-1)$
419 fi0=-mu*delta_0*r_2*%sinw[w]/sqrt(%Delta_0[r_1,r_2,w])^3//
420     *(3*(delta_0*r_1+r_2*%cosw[w])*delta_0*r_1/%Delta_0[r_1,r_2,w]-1)$
421 D2S=%D2S[fi1+fi0,fkdxx,0,0]$
422 return(D2S)$
423 };
424
425 macro D2Sr2r2[r_1,r_2,w]{
426 private D2S,fk,fi1,fi0,fdxdx,chi;
427 chi=%chi[r_1,r_2,%Dw[w]]$
428 fk=2*mu_2*B_2/r_2^3+B_2*chi^2//
429     *(B_1*r_1^2-3*B_2*r_2^2)/(B_1*r_1^2+B_2*r_2^2)$
430 fi1=-2*mu_2*B_2/r_2^3+mu_2*B_2/sqrt(%Delta_1[r_1,r_2,w])^3//
431     *(2-3*delta_1^2*r_1^2*%sinw[w]^2/%Delta_1[r_1,r_2,w])$
432 fi0=mu/sqrt(%Delta_0[r_1,r_2,w])^3//
433     *(2-3*delta_0^2*r_1^2*%sinw[w]^2/%Delta_0[r_1,r_2,w])$
434 fdxdx=B_2$convcf(fdxdx);
```

```
D2S=%D2S[fk+fi1+fi0,0,0,fdxdx]$
return(D2S)$
};

macro D2Sr2w[r_1,r_2,w]{
private D2S,fkdxx,fi1,fi0,chi;
chi=%chi[r_1,r_2,%Dw[w]]$
fkdxx=2*B_1*r_1^2*B_2*r_2/(B_1*r_1^2+B_2*r_2^2)*chi$
fi1=-mu_2*B_2*delta_1*r_1*%sinw[w]/sqrt(%Delta_1[r_1,r_2,w])^3//
    *(3*(r_2+delta_1*r_1*%cosw[w])*r_2/%Delta_1[r_1,r_2,w]-1)$
fi0=-mu*delta_0*r_1*%sinw[w]/sqrt(%Delta_0[r_1,r_2,w])^3//
    *(3*(r_2+delta_0*r_1*%cosw[w])*r_2/%Delta_0[r_1,r_2,w]-1)$
D2S=%D2S[fi1+fi0,fkdxx,0,0]$
return(D2S)$
};

macro D2Sww[r_1,r_2,w]{
private D2S,fkdxdx,fi1,fi0;
fkdxdx=B_1*r_1^2*B_2*r_2^2/(B_1*r_1^2+B_2*r_2^2)$
fi1=mu_2*B_2*delta_1*r_1*r_2/sqrt(%Delta_1[r_1,r_2,w])^3//
    *(3*delta_1*r_1*r_2*%sinw[w]^2/%Delta_1[r_1,r_2,w]+%cosw[w])$
fi0=mu*delta_0*r_1*r_2/sqrt(%Delta_0[r_1,r_2,w])^3//
    *(3*delta_0*r_1*r_2*%sinw[w]^2/%Delta_0[r_1,r_2,w]+%cosw[w])$
D2S=%D2S[fi1+fi0,0,0,fkdxdx]$
return(D2S)$
};


//functions for the derivatives of the Lagrangian
macro Delta_0[r_1,r_2,w]{
return(r_2^2+delta_0^2*r_1^2+2*delta_0*r_1*r_2*%cosw[w])$
};
macro Delta_1[r_1,r_2,w]{
return(r_2^2+delta_1^2*r_1^2+2*delta_1*r_1*r_2*%cosw[w])$
};

macro xi[r_1,r_2,dw]{
return((dw*B_2*r_2^2+G)/(B_1*r_1^2+B_2*r_2^2))$
};
macro chi[r_1,r_2,dw]{
return((dw*B_1*r_1^2-G)/(B_1*r_1^2+B_2*r_2^2))$
};


//gradient of the action from the gradient of the Lagrangian
macro DS[DLx,DLdx]{
private coefsDSx,coefsDSdx,omk,vk,complex,DS;
//Fourier coefficients
coefsDSx=%FourDecomp[DLx]$
```

```
484 coefsDSdx=%FourDecomp[DLdx]$
485 //vector with the term in omega*k
486 vnumR omk;
487 resize(omk,2*Nku);
488 for j=1 to Nku{
489 //order
490   vk=%orderFromVect[j]$
491 //the term in DLdx
492   complex=2*%scalar[omega,vk]*coefsDSdx[j]$
493   omk[j]=imag(complex)$
494   omk[j+Nku]=-real(complex)$
495   };
496 //gradient of the action
497 DS=vnumR[2*real(coefsDSx)+omk[1:Nku]:2*imag(coefsDSx)+omk[Nku+1:]]$
498 //correct the term in a_0
499 DS[1]=DS[1]/2$
500 //correct the term b_0
501 DS[Nku+1]=0$
502 return(DS)$
503 };
504
505 //approximate Hessian, for use with %Hinit
506 macro D2S[D2Lxx,D2Ldxx,D2Lxdx,D2Ldxdx]{
507 private D2S,D2Skpl,coefsD2Sxx,coefsD2Sdxx,coefsD2Sxdx,coefsD2Sdxdx//
508     ,vk,omk,kml;
509 //the approximate Hessian
510 vnumR D2S[1:2,1:2]$
511 resize(D2S,Nku,0);
512 //Fourier coefficients
513 coefsD2Sxx=%FourDecomp[D2Lxx]$
514 coefsD2Sdxx=%FourDecomp[D2Ldxx]$
515 coefsD2Sxdx=%FourDecomp[D2Lxdx]$
516 coefsD2Sdxdx=%FourDecomp[D2Ldxdx]$
517 //the index corresponding to k=0.
518 kml=%piv[0,0,0]$
519 for k=1 to Nku{
520 //order
521   vk=%orderFromVect[k]$
522 //omega*k
523   omk=%scalar[omega,vk]$
524 //D2Saa
525   D2S[1,1][k]=2*real(coefsD2Sxx[kml]
526       +omk*omk*coefsD2Sdxdx[kml])$
527 //D2Sab
528   D2S[1,2][k]=2*real(omk*coefsD2Sdxx[kml]//
529       -omk*coefsD2Sxdx[kml])$
530 //D2Sba
531   D2S[2,1][k]=-D2S[1,2][k]$
532 //D2Sbb
```

```
533     D2S[2,2][k]=D2S[1,1][k]$
534     };
535   //correct the terms in a_0 and b_0.
536   D2S[1,1][1]=D2S[1,1][1]/2$
537   D2S[1,2][1]=0$
538   D2S[2,1][1]=0$
539   D2S[2,2][1]=0$
540   return(D2S)$
541   };
542
543
544
545   //various useful tools
546
547   //D_\omega derivation operator
548   macro Dw[x]{
549   return(omega[1]*deriv(x,M_1)+omega[2]*deriv(x,M_2)+omega[3]*deriv(x,M_3))$
550   };
551
552   //cosine and sine for an angle with a linear term
553   macro cosw[w]{
554   return(cos(%cancelLinTerm[w])*cos(M[1]*W[1]+M[2]*W[2]+M[3]*W[3])//
555       -sin(%cancelLinTerm[w])*sin(M[1]*W[1]+M[2]*W[2]+M[3]*W[3]))$
556   };
557
558   macro sinw[w]{
559   return(sin(%cancelLinTerm[w])*cos(M[1]*W[1]+M[2]*W[2]+M[3]*W[3])//
560       +cos(%cancelLinTerm[w])*sin(M[1]*W[1]+M[2]*W[2]+M[3]*W[3]))$
561   };
562
563   //position of the harmonic (k_1,k_2,k_3) in a vector
564   macro piv[k_1,k_2,k_3]{
565   return(k_1*(2*kMax+1)^2+k_2*(2*kMax+1)+(k_3+1))$
566   };
567   //order of the harmonic from its position in a vector
568   macro orderFromVect[j]{
569   private k_1,k_2,k_3,N,ret;
570   N=2*kMax+1$
571   k_3=mod(j+kMax-1,N)-kMax$
572   k_2=mod(nint((j-k_3-1)/N+kMax),N)-kMax$
573   k_1=nint((j-(k_3+1)-k_2*N)/N^2)$
574   ret=vnumR[k_1:k_2:k_3]$
575   return(ret)$
576   };
577   //same as %piv but with a vector of three integers as input
578   macro pivV[vk]{
579   return(vk[1]*(2*kMax+1)^2+vk[2]*(2*kMax+1)+(vk[3]+1))$
580   };
581
```

```
582 //Fourier decomposition
583 macro FourDecomp[f]{
584 private ff,fcoefs,ffexp;
585 vnumC ff;
586 resize(ff,Nku)$
587 cfcoef_tabexp(f,fcoefs,ffexp)$
588 //fcoefs is a vector with the coefficients corresponding to the
589 //exponents of M_j in ffexp[j]
590 for l=1 to size(fcoefs) {
591 //only store the positive k's
592   if((ffexp[1][l]>0)||((ffexp[1][l]==0)&&((ffexp[2][l]>0)||//
593       ((ffexp[2][l]==0)&&(ffexp[3][l]>=0)))))then{
594     ff[%piv[ffexp[1][l],ffexp[2][l],ffexp[3][l]]]=fcoefs[l]$
595     };
596   };
597 return(ff)$
598 };
599
600 //convert a Fourier series to a vector
601 macro SeriesToVect[f]{
602 private ff,fcoefs,ffexp,fourdecomp;
603 vnumR ff;
604 resize(ff,2*Nku)$
605 fourdecomp=%FourDecomp[f]$
606 ff[:Nku]=real(fourdecomp)$
607 ff[Nku+1:]=imag(fourdecomp)$
608 return(ff)$
609 };
610 //for use with angles: ignores the linear term
611 macro SeriesToVectW[f]{
612 return(%SeriesToVect[%cancelLinTerm[f]])$
613 };
614
615 //convert a vector to a Fourier series
616 macro VectToSeries[ff]{
617 private f,vk;
618 f=ff[1]$
619 for j=2 to Nku{
620   if(j!=jK)then{
621     vk=%orderFromVect[j]$
622     f=f+(ff[j]+I*ff[j+Nku])*X_1^(vk[1])*X_2^(vk[2])*X_3^(vk[3])//
623       +(ff[j]-I*ff[j+Nku])*X_1^(-vk[1])*X_2^(-vk[2])*X_3^(-vk[3])$
624     };
625   };
626 return(f)$
627 };
628 //for use with angles: adds a linear term
629 macro VectToSeriesW[ff]{
630 return(%VectToSeries[ff]+W[1]*M[1]+W[2]*M[2]+W[3]*M[3])$
```

```
631 };
632
633 //cancels the linear term from a series
634 macro cancelLinTerm[ff]{
635 private fcoefs,ffexp;
636 coef_tabexp(ff,fcoefs,ffexp,M_1,M_2,M_3);
637 return(fcoefs[1])$
638 };
639
640
641 //scalar product
642 macro scalar[x,y]{
643 return(sum(x*y))$
644 };
645
646 //matrix product of a square matrix with a column vector
647 macro matProd[M,x]{
648 private ret;
649 vnumR ret$
650 resize(ret,size(M))$
651 for n=1 to size(M){
652   ret[n]=%scalar[M[n],x]$
653   };
654 return(ret)$
655 };
656
657 //exterior product of vectors
658 macro extProd[x,y]{
659 private ret;
660 vnumR ret[1:size(x)]$
661 for n=1 to size(x){
662   ret[n]=x[n]*y$
663   };
664 return(ret)$
665 };
666
667
668 //norm of a Fourier transformable function: \sum |c_k|
669 macro norm[u]{
670 private nor,ucoef,uexp;
671 cfcoef_tabexp(u,ucoef,uexp)$
672 nor=sum(abs(ucoef))$
673 return(nor)$
674 };
675
676
677 //to keep on going until Nfin iterations
678 macro keepOnGoing[Nfin]{
679 private Nold;
```

```
680 rn_1=%resizeDim[rn_1,Nfin]$
681 rn_2=%resizeDim[rn_2,Nfin]$
682 wn=%resizeDim[wn,Nfin]$
683 DSn=%resizeVect[DSn,Nfin]$
684
685 Nold=Niter$
686 Niter=Nfin$
687 %iter[Nold+1,Nfin]$
688 };
689
690 //resize a numerical vector
691 macro resizeVect[v,ns]{
692 private vp;
693 vnumR vp$
694 resize(vp,ns)$
695 vp[1:size(v)]=v$
696 return(vp)$
697 };
698 //resize a vector of series
699 macro resizeDim[v,ns]{
700 private vp;
701 dim vp[0:ns]$
702 vp[0:size(v)-1]=v$
703 return(vp)$
704 };
```

P4. Block inversion

The code for the program that inverts a matrix made of 36 diagonal blocks. We only show the beginning and end of the 8 000 line long program to give an idea of what it contains.

blockInverse6.t:

```
/*
Provides a macro to compute the inverse of a matrix made
of 36 diagonal blocks. The matrix is represented by a degree three
tensor: M[i,j][k] gives the diagonal element (k,k) of the block
(i,j) for i and j between 1 and 6.
The macro was computed using Mathematica.

I.Jauslin - last modified 26/04/2012
*/

//inverts the block matrix
macro inverseBlock[M]{
private Inv,
aa,ab,ac,ad,ae,af,
ba,bb,bc,bd,be,bf,
ca,cb,cc,cd,ce,cf,
da,db,dc,dd,de,df,
ea,eb,ec,ed,ee,ef,
fa,fb,fc,fd,fe,ff;

//declare the inverse
vnumR Inv[1:6,1:6];
//the blocks
aa=M[1,1]$ab=M[1,2]$ac=M[1,3]$ad=M[1,4]$ae=M[1,5]$af=M[1,6]$
ba=M[2,1]$bb=M[2,2]$bc=M[2,3]$bd=M[2,4]$be=M[2,5]$bf=M[2,6]$
ca=M[3,1]$cb=M[3,2]$cc=M[3,3]$cd=M[3,4]$ce=M[3,5]$cf=M[3,6]$
da=M[4,1]$db=M[4,2]$dc=M[4,3]$dd=M[4,4]$de=M[4,5]$df=M[4,6]$
ea=M[5,1]$eb=M[5,2]$ec=M[5,3]$ed=M[5,4]$ee=M[5,5]$ef=M[5,6]$
fa=M[6,1]$fb=M[6,2]$fc=M[6,3]$fd=M[6,4]$fe=M[6,5]$ff=M[6,6]$

Inv[1,1]=(bf*ce*dd*ec*fb - be*cf*dd*ec*fb - bf*cd*de*ec*fb + bd*cf*de*ec*fb +
    be*cd*df*ec*fb - bd*ce*df*ec*fb - bf*ce*dc*ed*fb + be*cf*dc*ed*fb +
    bf*cc*de*ed*fb - bc*cf*de*ed*fb - be*cc*df*ed*fb + bc*ce*df*ed*fb +
    bf*cd*dc*ee*fb - bd*cf*dc*ee*fb - bf*cc*dd*ee*fb + bc*cf*dd*ee*fb +
    bd*cc*df*ee*fb - bc*cd*df*ee*fb - be*cd*dc*ef*fb + bd*ce*dc*ef*fb +
    be*cc*dd*ef*fb - bc*ce*dd*ef*fb - bd*cc*de*ef*fb + bc*cd*de*ef*fb -
```

etc...

```
7981        ac*bb*cd*da*ee - ab*bc*cd*da*ee - ad*bc*ca*db*ee + ac*bd*ca*db*ee +
7982        ad*ba*cc*db*ee - aa*bd*cc*db*ee - ac*ba*cd*db*ee + aa*bc*cd*db*ee +
7983        ad*bb*ca*dc*ee - ab*bd*ca*dc*ee - ad*ba*cb*dc*ee + aa*bd*cb*dc*ee +
```

```
7984          ab*ba*cd*dc*ee - aa*bb*cd*dc*ee - ac*bb*ca*dd*ee + ab*bc*ca*dd*ee +
7985          ac*ba*cb*dd*ee - aa*bc*cb*dd*ee - ab*ba*cc*dd*ee + aa*bb*cc*dd*ee)*ff)
7986  $
7987
7988  return(Inv)$
7989  };
```

P5. Conjugate gradient - numerical instability

The code for the program that shows the numerical instability of the conjugate gradient algorithm.

conjGradTest.t:

```
/*
Test of the conjugate gradient method.
The macro testInverse computes a random symmetric matrix A
and a random vector b, computes x such that Ax=b, and returns
  ||Ax-b||/N
where N is the size of b.
To run the algorithm execute testInverse, for example
  %testInverse[20,1,100];

I.Jauslin - last modified 27/04/2012
*/

//environment

//quadruple precision floats
_modenum=NUMQUAD$
//neglect anything below the given precision
_cleaneps=1e-32$
_cleanflag=1$


//tests the inversion procedure
//generates a random symmetric matrix A and a random vector b
//nn is the size of the vectors and matrices
//nMax is the largest value allowed in the random matrix
//nrand is the number of integers the random number generator chooses from
macro testInverse[nn,nMax,nrand]{
vnumR A[1:nn]$
resize(A,nn)$
vnumR b$
resize(b,nn);
for j=1 to nn{
  A[j][j]=((nrand-1)/2-random(nrand-1))*2*nMax/(nrand-1)$
  for k=j+1 to nn{
    A[j][k]=((nrand-1)/2-random(nrand-1))*2*nMax/(nrand-1)$
    A[k][j]=A[j][k]$
    };
  b[j]=((nrand-1)/2-random(nrand-1))*2*nMax/(nrand-1)$
  };
x=%inverse[A,b,b]$
return(sum(abs(%matProd[A,x]-b))/size(b))$
};
```

```
43
44
45  //gives the (exact) solution of Ax=b
46  //using an iterative algorithm starting from an arbitrary x0
47  macro inverse[A,b,x0]{
48  private g,h,lambda,gamma,ng,x,ah;
49  g=%matProd[A,x0]-b$
50  h=g$
51  x=x0$
52  n=1$
53  //stop the algorithm if the right x has been reached
54  while((n<=size(b)) && (sum(abs(%matProd[A,x]-b))>1E-15)) do {
55    ah=%matProd[A,h]$
56    lambda=%scalar[g,h]/%scalar[h,ah]$
57    x=x-lambda*h$
58    ng=g-lambda*ah$
59    gamma=%scalar[ng,ng]/%scalar[g,g]$
60    g=ng$
61    h=g+gamma*h$
62    n=n+1$
63    };
64  msg("last step: %d/%d\n",n-1,size(b))$
65  return(x)$
66  };
67
68
69  //scalar product
70  macro scalar[x,y]{
71  return(sum(x*y))$
72  };
73  //matrix product of a square matrix with a column vector
74  macro matProd[M,x]{
75  private ret;
76  vnumR ret$
77  resize(ret,size(M))$
78  for n=1 to size(M){
79    ret[n]=%scalar[M[n],x]$
80    };
81  return(ret)$
82  };
```

References

[Ar63a] V.I. Arnol'd - *Proof of a theorem of A.N. Kolmogorov on the preservation of conditionally periodic motions under a small perturbation of the Hamiltonian*, Uspehi Matematicheskih Nauk, Vol. 18, p. 13-40, english translation: Russian mathematical surveys, Vol. 18, n. 5, p. 9-36, 1963.

[Ar63b] V.I. Arnol'd - *Small denominators and problems of stability of motion in classical and celestial mechanics*, Uspehi Matematicheskih Nauk, Vol. 18, p.91-196, english translation: Russian mathematical surveys, Vol. 18, p.85-193, 1963.

[Ar78] V.I. Arnol'd - *Mathematical methods of classical mechanics*, Springer, 1978.

[Ar88] V.I. Arnol'd - *Dynamical systems III*, in the series *Encyclopædia of Mathematical Sciences*, Vol. 3, edited by R.V. Gamkrelidze, Springer, 1988.

[CC97] A. Celletti, L. Chierchia - *On the stability of realistic three-body problems*, Communications in Mathematical Physics, Vol. 186, p. 413-449, 1997.

[CC98] A. Celletti, L. Chierchia - *KAM stability estimates in celestial mechanics*, Planetary and Space Science, Vol. 46, n. 11-12, p. 1433-1440, 1998.

[Ga94] G. Gallavotti - *Twistless KAM tori*, Communications in Mathematical Physics, Vol. 164, pp145-156, 1994.

[Ga04] G. Gallavotti, F. Bonetto, G. Gentile - *Aspects of Ergodic, Qualitative and Statistical Theory of Motion*, Springer, 2004.

[GL10] M. Gastineau, J. Laskar - TRIP*: a computer algebra system dedicated to celestial mechanics and perturbation series*, ISSAC 2010 Software Presentations, 2010.

[GL12] M. Gastineau, J. Laskar - TRIP 1.2a4, IMCCE, Observatoire de Paris, 2012.

[Ko54] A.N. Kolmogorov - *Preservation of conditionally periodic movements with small change in the Hamilton function*, Doklady Akademii Nauk S.S.S.R., Vol. 98, p. 527-530, 1954.

[La89] J. Laskar - *A numerical experiment on the chaotic behavior of the Solar System*, Nature, Vol. 338, p. 237-238, 1989.

[La99] J. Laskar - *Introduction to frequency map analysis*, Proceedings of NATO ASI, Hamiltonian systems with three or more degrees of freedom, p. 134-150, 1999.

[LG09] J. Laskar, M. Gastineau - *Existence of collisional trajectories of Mercury, Mars and Venus with the Earth*, Nature, Vol. 459, 817-819, 2009.

[La10] J. Laskar - *Le Système Solaire est-il stable?*, Séminaire Poincaré XIV, p. 221-246, 2010.

[LG05] U. Locatelli, A. Giorgilli - *Construction of Kolmogorov's normal form for a planetary system*, Regular and Chaotic Dynamics, Vol. 10, n. 2, p. 153-171, 2005.

[Ma82a] J.N. Mather - *Concavity of the Lagrangian for quasi-periodic orbits*, Comentarii Mathematici Helvetici, Vol. 57, n. 1, p. 356-376, 1982.

[Ma82b] J.N. Mather - *Existence of quasi-periodic orbits for twist homeomorphisms of the annulus*, Topology, Vol. 21, n. 4, p. 457-467, 1982.

[Mo62] J. Moser - *On invariant curves of area-preserving mappings of an annulus*, Nachrichten der Akademie der Wissenschaften zu Göttingen Mathematisch-Physikalische Klasse, p. 1-20, 1962.

[Mo86] J. Moser - *Minimal solutions of variational problems on a torus*, Annales de l'IHP, section C, Vol. 3, n. 3, p. 229-272, 1986.

[Mo88] J. Moser - *A stability theorem for minimal foliations on a torus*, Ergodic theory and Dynamical systems, Vol. 8* (Charles Conley memorial issue), p. 251-281, 1988.

[NR] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B. P. Flannery - *Numerical Recipes, the art of scientific computing*, third edition, Cambridge University Press, 2008.

[Pe79] I.C. Percival - *A variational principle for invariant tori of fixed frequency*, Journal of Physics A, Vol. 12, n. 3, L57, 1979.

[Po71] E. Polak - *Computational methods in optimization*, Academic Press, 1971.

[Ro95] P. Robutel - *Stability of the planetary three-body problem - II. KAM theory and existence of quasiperiodic motions*, Celestial Mechanics and Dynamical Astronomy, Vol. 62, n. 3, p. 219-261, 1995.

[SZ89] D. Salamon, E. Zehnder - *KAM theory in configuration space*, Commentarii Mathematici Helvetici, Vol. 64, p. 84-132, 1989.