A new subdivision algorithm for the flow propagation using polynomial algebras

Daniel Pérez-Palau

CNES, 18 avenue Edouard Belin, 31401 Toulouse

Gerard Gómez

Universitat de Barcelona & IEEC, Gran Via 585, 08007 Barcelona

Josep J. Masdemont

Universitat Politècnica de Catalunya & IEEC, Diagonal 647, 08028 Barcelona

Abstract

The Jet Transport method has emerged as a powerful tool for the numerical integration of ordinary differential equations; it uses polynomial expansions to approximate the flow map associated to the differential equation in the neighborhood of a reference solution. One of the main drawbacks of the method is that the region of accuracy becomes smaller along the integration. In this paper we introduce a procedure to determine a ball covering the set of given initial conditions that keeps the accuracy of the integration within a selected threshold. The paper gives detailed explanations of the algorithm illustrated with some examples of applicability, as well as a comparison with a previous existing method for the same purpose.

Keywords: Jet Transport; polynomial algebra; subdivision algorithms; ODE polynomial propagation

2010 MSC: 10.070

1. Introduction

Let $\dot{x} = f(t, x)$, $x \in \mathbb{R}^n$, be a system of ordinary differential equations (ODE), and $\phi_t(t_0, x_0)$ the associated flow map: if $x(t)$ denotes the solution of the ODE such that $x(t_0) = x_0$, then $x(t) = \phi_t(t_0, x_0)$. The Jet Transport (JT), also known as Differential Algebra, procedure is a semi-numerical method that propagates a neighbourhood U of x_0 instead of the single initial condition x_0; this is, at the first time step h of the propagation, the initial condition x_0 is replaced by a polynomial of degree one $P_{t_0,x_0}(\xi) = x_0 + \xi \in \mathbb{R}^n$ that parameterises U, and a higher degree polynomial approximation $P_{t_0+h,x_0}(\xi)$ of $\phi_{t_0+h}(t_0, x_0+\xi)$ is computed. This resulting polynomial is propagated in the next step, and the procedure is repeated recursively. The basic idea of the method is shown schematically in Fig. 1.

The propagated polynomials $P_{t,x}(\xi)$ are computed using an implementation of a numerical integration method for ODEs in which the real number floating point arithmetic is replaced by a polynomial algebra (i.e.

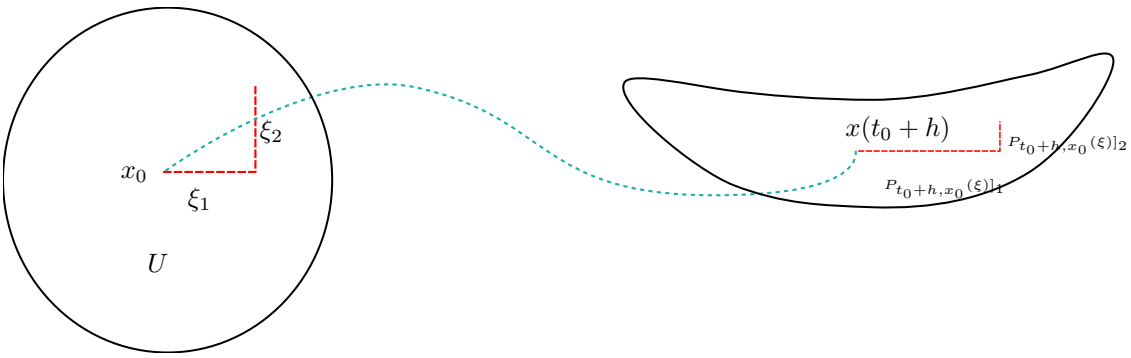

Figure 1: Schematic idea of the JT propagation procedure. The circle on the left represents the neighbourhood U parameterised by (ξ_1, ξ_2), and the right hand side ellipse is its image at time $t_0 + h$. In the figure $x(t_0 + h)$ stands for $\phi_{t_0+h}(t_0, x_0)$, and the image of $x_0 + (\xi_1, \xi_2) \in U$ is the point $P_{t_0+h,x_0}(\xi) = x(t_0 + h) + (P_{t_0+h,x_0}(\xi)\,|_1, P_{t_0+h,x_0}(\xi)\,|_2)$.

all the arithmetic operations are done using truncated polynomials up to a certain degree). The polynomials $P_{t,x}(\xi)$ provide, up to a certain order, the solutions of the variational equations associated to the ODE without writing and integrating them explicitly. Therefore, the only tools that are needed are: a numerical integration method for ODEs and a polynomial algebra package.

For common numerical integration methods, such as Runge-Kutta, Taylor or symplectic, the step-size selection is done according to a local truncation error estimate. For the JT procedures it is also necessary to control the size of U. This is the problem we address in the present paper. In the proposed approach, whenever necessary, a subdivision strategy is applied either increasing the number of polynomials or splitting regions for the propagation.

Jet Transport methods where introduced by M. Berz in 1986 ([7]) to study beam dynamics in particle accelerators problems and, since then, have been used in several other fields, for instance: in Astrodynamics to study the close approaches of Near Earth Asteroids ([3, 4]), to compute a Gaussian particle filter for spacecraft navigation ([18]), or to detect structures in dynamical systems using several indicators ([14]). JT methods have also been used to compute the time evolution of probability density functions (PDF) according to the flow associated to an ordinary differential equation ([13, 21, 5]).

Currently, there are several implementations of the Jet Transport/Differential Algebra methods. Some of the most well known are: COSY INFINITY (M. Berz and K. Makino [10]); TIDES (A. Abad et al. [1]); CAPD DynSys Library (D. Wilczak [19]), and DACE, which is an implementation developed from COSY specially adapted to the space community (Rasotto et al. [15]). For this work we have used our own one (D. Pérez-Palau [13]), which includes all the basic algebra and functions needed for the implementation of the numerical integration method, as well as some additional ones such as the inversion of functions and an equation solver.

One of the main drawbacks of the JT methods is that the region where their accuracy is below a certain threshold becomes smaller along the integration. This paper presents a new procedure to determine how and when the region and associated polynomials can be split to maintain the required accuracy. The method is based on the covering of the propagated states by new neighbourhoods. The algorithm presented contains some ideas similar to the ones of the interval enclosure methods [6, 2], in which the results of the operations are assured to be in a given interval. In the current algorithm, the results of the operations are given by polynomials, which are assured to be in a given region. The new algorithm is compared with the Automatic Domain Splitting (ADS) approach, an algorithm developed by A. Wittig et al. [20] for this same problem, in which the basic idea is the division and rescaling of the propagation polynomials along the integration.

It must be noted that both approaches mentioned have two main steps: the first one detects when the splitting should be done, and the second accounts for how the division of neighbourhoods or polynomials is implemented. In both cases the procedures work schematically as follows. First a certain neighbourhood is propagated until some condition breaks because the accuracy of the propagation is below some fixed tolerance. Up to that point the method is a usual JT flow propagation. When the condition breaks the second part of the algorithm starts: the polynomial or the set of points, depending on the selected strategy, splits. After the division, the usual Jet Transport flow propagation is started with new initial conditions.

The paper is organised as follows: in Section 2 we briefly introduce the step control strategy for the integrator used as well as how to determine if the polynomials are accurate enough; in Section 3 we review the (ADS) approach; Section 4 describes the procedure that we have developed; in Section 5 we give some numerical tests and comparisons between both methods, while the last section ends with some remarks and conclusions.

2. Step-size control in polynomial algebra propagation

In this section we briefly discuss how to determine the step size in a JT procedure. As numerical integration method we have used Taylor's method for ODE (as implemented by Jorba and Zou [9]). Given a certain accuracy level ε, the optimal step of this method is given by (see [16] for details)

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

where $x^{(i)}$ are the coefficients of Taylor's method solution written in powers of the step size h at each step, this is:

$$\phi_{t_n+h}(t_n, x_n) = \sum_{i=0}^{n} x^{(i)}(t_n, x_n) h^i.$$

In our case, the coefficients $x^{(i)}$ are polynomials, so for each coordinate x_m we have:

$$x_m^{(i)} = \sum_k c_{m,k}^{(i)} \xi^k,$$

and the step size control must be done using the coefficients $c_{m,k}^{(i)}$. Denoting by

$$\psi_k^i = \max_{1\leq m\leq n} |c_{m,k}^{(i)}|,$$

an approximation of the optimal value of h, in order to control all the orders of the expansion, is

$$h_{opt} = \min_k \left\{ \left(\frac{\varepsilon e^2 \psi_k^1}{\psi_k^{n-1}} \right)^{\frac{1}{n-2}}, \left(\frac{\varepsilon \psi_k^1}{\psi_k^n} \right)^{\frac{1}{n-1}} \right\}.$$

The main idea behind such approximation is based on the fact that each coefficient $c_{m,k}^{(i)}$ of the expansion can be understood as the k-th coefficient in the phase space variables and the i-th coefficient in the time variables, all of them associated to the Taylor expansion of the m-th component. Under such situation, consider the maximum of $c_{m,k}^{(i)}$ is equivalent to considering the polynomial as the expanded solution of the ODE including all the high order variational equations.

As it has already been said, in the JT method the size of the set U that can be propagated with a given accuracy decreases as the integration time increases. As in the step-size control of Taylor's method, there will be also a domain, to be estimated, where the JT resulting polynomial will have an error below a given tolerance. Ideally, it would be convenient to know the value of the maximum size of the domain that can be propagated keeping the above error condition satisfied before doing any further time-step of the procedure. However, it is not known how to do this a priori estimation. The size of the domain U can only be estimated after each integration step, and this will be done in an analogous manner as the step-size control selection of Taylor's method, now using the higher order terms on the resulting polynomial.

Assume that at a certain time t the JT polynomial is,

$$P_{t_0+t,x_0}(\xi) = \sum_{\substack{1\leq m\leq n \\ |k|\leq n}} a_{m,k}\xi^k.$$

A straightforward option is to select the size of U in such way that

$$a_{m,k}\xi^k \leq \varepsilon_{JT},$$

where ε_{JT} is a given tolerance. In this way, the last term is not adding a contribution larger than ε_{JT} to the polynomial. In case where we require U to have the same size in any coordinate direction we get,

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

In case that the last term were zero, one can use an exponential fit of the size of all the known non-zero coefficients, up to a certain order, to obtain an accurate estimate of the size of the next order (see [20]).

Hence, if we want to propagate a given region U of the phase space, we fix an initial radius r_0 and an initial condition $\vec{x}_0$, in such way that U is contained in the ball of radius r_0 and centre x_0 and start the

propagation. Whenever we detect that, according to the previous criterion, $P(\xi)$, with $\xi \in U$, is not a good enough approximation of the flow, we proceed with a subdivision strategy, either using two or more polynomials or using two or more regions for the propagation.

3. Splitting the polynomial

In the ADS method introduced in [20] the polynomial approximation of the flow is split in two new ones whenever some accuracy condition is not fulfilled. In this method the initial neighbourhood is parameterised by the n-dimensional parallelogram $[-1,1]^n$. At each step of the integration, once the polynomial approximation $P(\xi)$ of the flow is available, the following test is performed:

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

Whenever the test fails, the following algorithm is applied to determine the splitting of $P(\xi)$.

- Look for the more expansive direction e_i, again using an exponential fit for the coefficients of the polynomials in each variable x_j, and taking i as the value giving the largest error. The splitting is only performed in the i direction.
- Split $P(\xi)$ in two polynomials $P_\pm(\xi)$ defined by

$$P_\pm(\xi) = P\left(\xi_1, ..., \xi_{i-1}, \frac{\xi_i}{2} \pm \frac{e_i}{2}, \xi_{i+1}, ..., \xi_n\right),$$

  where $e_i = (0, ..., 0, 1, 0, ...0)$ is the i-the vector of the canonical base of $\mathbb{R}^n$. Each polynomial is defined again in $[-1,1]^n$, but these two boxes are now parameterisations of smaller regions.

After the splitting, the propagation continues for both P_+ and P_-. Taking rectangular boxes has several benefits. The first one is that the boxes do not overlap. In addition, once the splitting strategy is fixed, it is straightforward to use. The final result of the method consist in N polynomials. To determine the propagation for a given initial point x we only need to evaluate the appropriate polynomial at the transformed coordinates of x in the box $[-1,1]^n$. A possible drawback of the parameterisations is the following: the vertices of $[-1,1]^n$ are farther from the origin than other points; it can happen that if the vertices are well approximated, according to the tolerance test, there will be points outside the box that are also well approximated by the polynomial but cannot be propagated with the same polynomial.

In this procedure it should be also possible to use, instead of e_i, any other unitary direction c ($\|c\| = 1$) together with the polynomials $P_{\pm}(\xi) = P\left(\frac{\xi}{2} \pm \frac{c}{2}\right)$. However, using an arbitrary direction, the initial region is not completely covered by the two new polynomials $P_{\pm}(\xi)$, and also in other subregions they will overlap each other. These two facts make to consider splittings in arbitrary directions not suitable.

4. Splitting the neighbourhood

The method that has been developed splits the domains of integration instead of the polynomial approximation of the flow. The basic idea consists in propagating an initial neighbourhood until a certain time T, when an accuracy condition fails, and then, according to certain rules, we construct a new set of polynomials, which propagates points around $\phi_T(t_0, x_0)$.

To detect when the propagation fails, the *maximum initial box test*, determined by the value of ξ_{max}, discussed in (1), is used. At the beginning of the propagation a minimum initial box of size r_0 is fixed, this determines the neighbourhood U_0 that we want to propagate with enough accuracy. Therefore, we consider that the test fails whenever the size ξ_{max} of the maximum initial box is below the size r_0.

The next point to consider is how to generate new neighbourhoods U_1^i in order to cover the propagation of U_0. For its determination two different strategies have been implemented. The first one is a covering procedure that uses the information that we have in the propagated polynomial to determine where we must locate the new sets U_1^i. The second one uses tracer points, to be defined, in order to determine the regions that must be covered. In both cases new radii r_1^i are determined.

Along the steps of the algorithm there will be, in general, more than one neighborhood U_j^i. Each neighbourhood is defined by a polynomial that is used to compute the propagation. In this situation, the maximum initial box test must be checked for each one of the polynomials. If one of the polynomials breaks the accuracy condition, then the splitting procedure is applied to all of them. Observe that now the maximum initial box must be compared with the associated minimum box size r_0.

The following notation will be used: U will denote the neighbourhood of x_0 parameterised by $x_0 + \xi$, with $\|\xi\| < r_0$; $V = \phi_T(t_0, U)$ will be the image at time T of U computed using the flow associated to the differential equation, and $\tilde{V} = P(U)$ will denote the image at time T of U computed using the polynomial $P(\xi) = P_{t_0}^T(\xi) \approx \phi_T(t_0, x_0 + \xi)$. The maximum initial box test will fail if the maximum value ξ_{max} is below a minimum box size radius r_0. In general, if U is a ball around x_0, its image V will be an ellipsoidal (or close to it) neighbourhood. In the extremal case where it is well approximated by non-convex neighborhoods the algorithms presented are well adapted.

4.1. The covering procedure

The first splitting procedure uses the neighbourhood $\tilde{V}$, computed at a certain epoch T using $P(\xi)$, to define two new neighbourhoods, U_+ and U_-, covering V.

By means of $P(\xi)$, we look for a point $x_e = P(\xi_e)$ such that $\|v_e\| = \|P(\xi_e) - P(0)\|$ is maximum among all the points with $\|\xi_e\| = r_0$. The point x_e is obtained using Newton's method applied to the derivative of the distance function, that refines a first guess obtained after a preliminary exploration of a rough grid of points. Note that, since the polynomial $P(\xi)$ is available, Newton's method only requires the computation of derivatives of polynomials, which is done using the implemented polynomial algebra.

In the next step two new neighbourhoods $U_\pm$ are added. They are located along the most expanding direction determined by x_e, one at each side of the centre of $\tilde{V}$, and with their centers at $c_\pm = P(0) \pm v_e/2$, respectively.

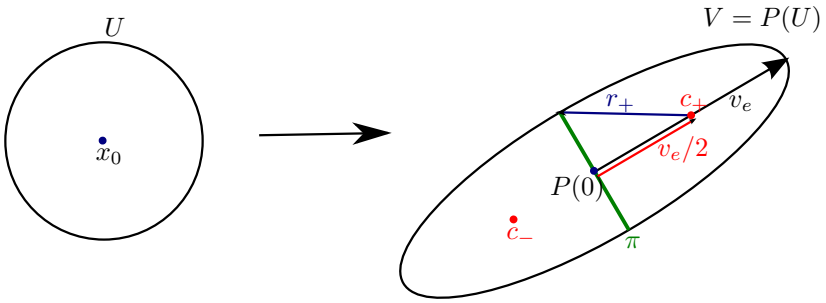

Figure 2: Schematic explanation of the covering procedure. The propagated neighbourhood V decomposes in two new ones $U_\pm$ with centres $c_\pm$ along the more expansive direction v_e and radius $r_+ = r_-$, such that V is completely covered by them.

Taking $r_\pm = \|v_e\|/2$ as the radii of both new neighbourhoods $U_\pm$ is not a good option, since then V will not be completely covered by them: there will be points in the hyper-plane π through $P(0)$ and perpendicular to v_e that will not be in any of the two new neighbourhoods. Using one of the two half neighbourhoods of V determined by π, the value of $r_\pm$ is taken as the maximum distance from the selected neighbourhood to the associated center c_+ or c_-. Again, a grid search is used to determine a good initial guess for $r_\pm$, that is refined using Newton's method applied to the derivative of the distance function. Figure 2 shows a scheme of the different ingredients that take part in this covering procedure.

The neighbourhoods $U_\pm$ are labelled as U_p^i, where p denotes the splitting step in which they are generated and i is an index for its identification. Figure 3 shows the results obtained with the above algorithm applied to propagate an initial neighbourhood with $r_0 = 0.1$ around a point located in the lower part of the circulation regime of the pendulum ($x'' = -\sin x$, $(x_0, x_0') = (0, -3.4)$).

Some of the problems of this splitting become clear looking at Figure 3. The most notorious one is that at the end of the propagation the covered region by the U_p^i sets is much larger than the one really needed (bounded by the black thick line in the figure). This is because at each step we are enlarging the propagated region. At a first glance, it seems that this would not be a major problem apart from the fact that in regular regions of the vectorfield the procedure would require a higher computational cost. However, if the integration has a close passage to a singularity over-covered by the algorithm, then the failure of the splitting

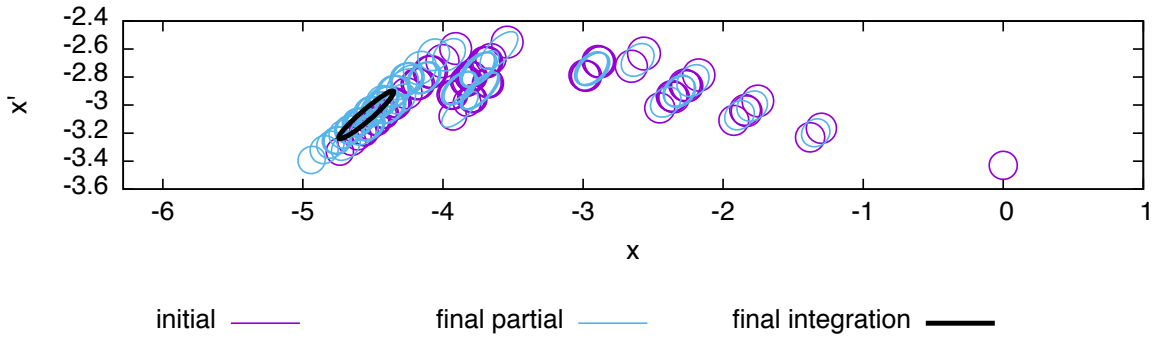

Figure 3: Evolution of the splittings of the subdivision algorithm for an initial ball around the point $(x, x') = (0, -3.4)$. The initial region is in the right lower part. The purple regions are the sets U_p^i, the blue regions are their propagations V_p^i. The black thick line on the left hand side surrounds the "true" final propagation of the initial neighbourhood, computed using a direct numerical integration.

160 condition will happen often and the new set to be covered will require more subsets, since it will be much more elongated.

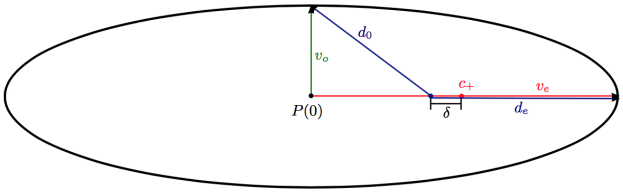

Figure 4: Schematic representation of the computation of the δ variation needed to match the distance of maximum expansion and the distance in the normal space.

In order to solve the above drawback we introduce the following modification. In addition to v_e, we search for a new direction v_o defined as the maximum expansion direction orthogonal to v_e. The centre of the new box is moved closer to $P(0)$ and its radius slightly reduced according to the following. Let δ denote the displacement of the centre as shown in Figure 4; the new location of the centre must satisfy

$$\frac{\|v_e\|}{2} + \delta = \sqrt{\left(\frac{\|v_e\|}{2} - \delta\right)^2 + v_o^2},$$

from which we get $\delta = \dfrac{\|v_o\|^2}{2\|v_e\|}$. Therefore, locating the centre of the new neighbourhood at

$$c_\pm = P(0) \pm \left(\frac{\|v_e\|}{2} - \frac{\|v_o\|^2}{2\|v_e\|}\right),$$

and taking as its radius

$$r_\pm = \frac{\|v_e\|}{2} + \frac{\|v_o\|^2}{2\|v_e\|},$$

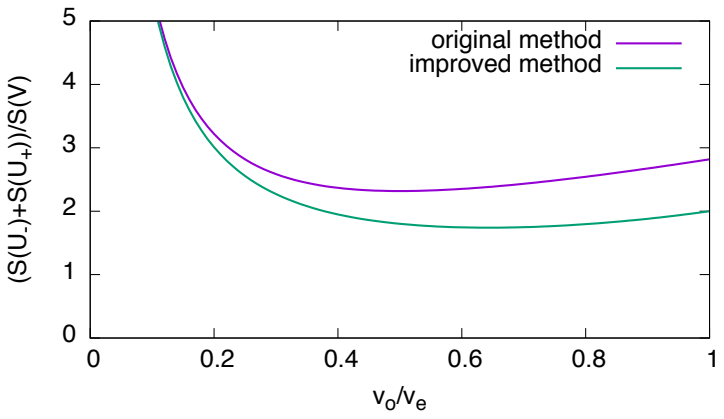

Figure 5: Ratio between the area covered by the new neighbourhoods U_+ and U_- over the area of an elliptical approximation of the neighborhood V as a function of the ratio between the length of the vectors v_e and v_o.

the new neighbourhood U_p^i is the smallest one that covers the corresponding part of V.

Figure 5 shows the ratio of the sum of the areas of U_+ and U_- over the area of the almost-elliptical neighborhood V as a function of the ratio $\|v_e\|/\|v_o\|$. It shows that the improvement reduces the overestimated area. For low elliptical ratios the improved method covers almost twice the area of V, due to the fact that both neighborhoods U_+ and U_- almost completely overlap. For those cases with high eccentricity the overestimation is large due to the elongated shape V. In addition, if V is far to be elliptical the approximation used to compute the overestimation is not valid.

Another problem that comes out from Figure 3 is the existence of neighbourhoods $U_\pm$ that almost completely overlap. This is already clear at the first step. The main reason of this behaviour is that the direction in which the propagation of the initial neighbourhood is expanded is almost the same for each iteration. Figure 6 shows the overlapping of the sets U_{+-} and U_{-+} at the second step of the method. To avoid this drawback, a safety factor ρ is introduced. Once all the new neighbourhoods U_p^i have been computed, and before the next propagation step, the distance between their centres is computed. If there are two centres whose distance is less than ρ times their radii, then two associated neighbourhoods are merged. The centre of the new neighbourhood is located at the middle of the two removed ones, and the radius is enlarged by one half of the distance between the two centres. The value of ρ must be tuned for each application; if it is too small the procedure does not merge any pair of neighborhoods, and if it is too large the procedure joins too many neighborhoods, and their size becomes too large to be propagated with enough accuracy. The value $\rho = 0.2$ has been used in most of the examples of this paper.

There is still some possible overlapping of the U_p^i when their centres are close but not close enough to be merged by the above improvement. In the case of highly non-convex sets the described method covers completely the set V, however the overestimation of the covering is big. For these reasons an alternative is

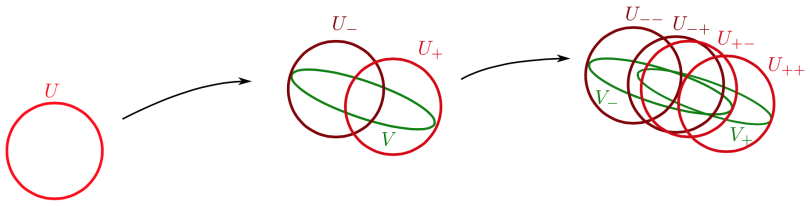

Figure 6: The neighbourhoods U_{+-} and U_{-+} almost complete overlap at the second step of the covering procedure.

required. The next algorithm presents a new way to set the neighbourhoods U_p^i; it uses the information provided by some points that are propagated up to the splitting time to determine the region V to be covered by the new U_p^i.

4.2. The use of tracers to control the covering

The following algorithm uses a set of points $\{\sigma_0^0, \sigma_0^1, \ldots, \sigma_0^m\}$ in U_0, or tracers, to get information about the evolution of U_0 under the flow associated to the dynamical system defined by $f(t, x)$. The tracers are used to identify, at each step p the number and the location of U_p^i sets.

For a given U_0, the associated tracers are located at its center, $\sigma_0^0 = x_0$, and its boundary, $\mathcal{T}_0 = \{\sigma_0^1, \ldots, \sigma_0^m\}$. In the 2-dimensional case, the boundary points will be $\sigma_0^i = x_0 + (r_0 \cos(2\pi i/m), r_0 \sin(2\pi i/m))$. The discussion for higher dimensions is given later. The inclusion of x_0 as a tracer is convenient since allows to control the central part of the neighborhood which, for usual applications, is the one with highest interest.

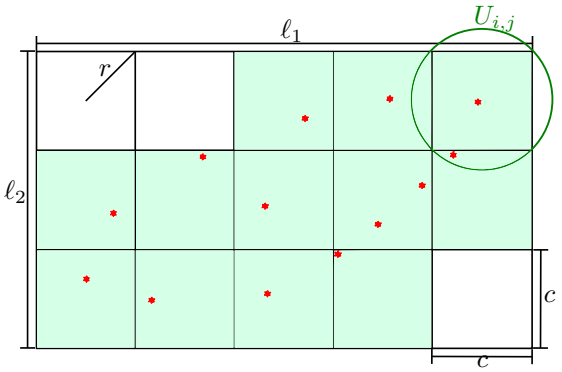

Figure 7: Elements that take part in the selection of the neighbourhoods U_p^i. The red dots are the tracers and the light green boxes the ones that will contain a new neighbourhood U_p^j.

All the tracers are propagated up to the splitting time using the JT: $\sigma_p^i = P_T(\sigma_{p-1}^i)$. Then, the new set of neighborhoods U_p^i is determined in such way that all the σ_p^i are covered by at least one of such neighbourhoods. Algorithm 1 (illustrated in Figure 7) is used to determine the new set U_p^i, and the number

of the polynomials to be propagated. In order to ensure the covering of the tracers, a grid of n-dimensional boxes covering the full set of tracers at time T is used. The neighborhoods U_p^i are located only at those boxes that contain a tracer. Observe that in the grid each box will have a multi-index k associated. The box with multi-index k is the one located in the k_i-th position in the i-th direction of the grid.

Algorithm 1. *Consider the n-dimension system $\dot{x} = f(t, x)$, $x \in \mathbb{R}^n$. At the p-th covering step:*

1. *For a given value r of the radii of the sets U_p^i, compute the length c of the side of a square (or, in general, of a hyper-cube) such that r is the length of its diagonal and so $c = \sqrt{4r^2/n}$.*
2. *Compute the minimum and maximum values of all the coordinates of the tracers $\sigma_p^1, \ldots, \sigma_p^m$:*

$$a_\tau = \min_{0 \le i \le m} \sigma_p^i|_\tau, \quad b_\tau = \max_{0 \le i \le m} \sigma_p^i|_\tau, \quad \tau = 1, ..., n.$$

   These two values will be used to determine the boundaries of the grid.
3. *Compute the grid size ℓ as: $\ell_\tau = \left(\left[\dfrac{b_\tau - a_\tau}{c} \right] + 1 \right) c$, where $[\,\cdot\,]$ denotes the integer part.*
4. *Fix an n-dimensional grid. Each side will have $\left[\dfrac{b_\tau - a_\tau}{c} \right] + 1$ boxes, and will start at $\tilde{a}_\tau = a_\tau - \dfrac{\ell_\tau - b_\tau + a_\tau}{2}$.*
5. *If a box with multi-index k contains a tracer t_p^i, associate to it a new neighbourhood U_p^i of radius r and center $a_i + (k_i + 0.5)\, c$.*
6. *A polynomial P_p^i is initialised for each U_p^i. If a given box contains two or more tracers, only one polynomial is initialised.*
7. *All the polynomials P_p^i are propagated together until one of them breaks the division condition or the final time is achieved.*

Observe that the grid size ℓ defined in step *3* is larger than the distance between the maximum and the minimum in each direction. This increases the region where the new sets are selected and, in this way, if the set of tracers does not reach the most distant point to the center, there are still chances to capture it.

The value of r for the new neighbourhoods must be selected for each problem after some preliminary tests. If it is too large the new polynomials will break the split condition after a short propagation time, and if it is too small there will appear too many polynomials and the algorithm will become slow. A compromise between both facts must be achieved.

Figures 8 and 9 show, for the simple pendulum, the elements introduced in the above algorithm as well as the first four steps of the procedure. Figure 8 shows, for the first step, how the selection of the U_1^i sets is done using the tracers, and the result of the propagation of U_0 and the U_1^i. The three left hand side plots of Figure 9 show the results of the second, third and fourth step of the splitting procedure. In all the plots, the purple curve is U_0 and its "true" propagation using Taylor's method, and the tracers are the green crosses,

which are propagated by $P(\xi)$ and used to set the grid of boxes (in blue) to select the new neighbourhoods U_1^i (in yellow). To obtain the figure the initial condition $x_0 = (0, -3.5)$ has been integrated using as a value of the radii for the new neighbourhoods $r = 0.1$ and a tolerance to split $\xi_{\max} = 0.035$.

The bottom left plot of Figure 9 corresponds to the fourth step of the procedure, when the final time at this step has been reached (the splitting condition is activated). There, we can see the result of the "true" integration of the differential equation (in purple) and images of the U_3^i sets (in green). It can be seen that the U_3^i sets do not cover completely the purple one. This happens because in the previous step not all the boxes containing the propagation of U_0 were selected. This is because the U_0 set expands with time and, therefore, the tracers in it will also expand and separate from each other.

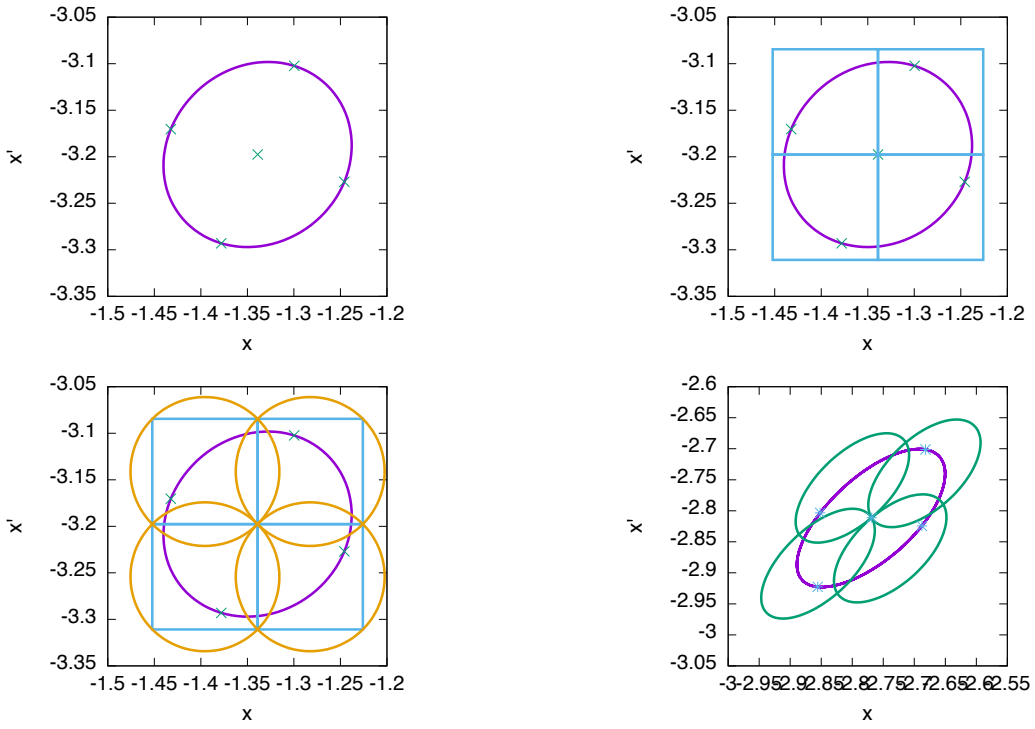

Figure 8: From left to right and from top to bottom, first steps of Algorithm 1 for an initial condition in the circulating regime of the simple pendulum. The purple curve is U_0 and its "true" propagation using Taylor's method, the tracers (green crosses) are propagated using $P(\xi)$. On the traces we set the grid of boxes (in blue) to select the new neighbourhoods U_1^i that are propagated (in yellow). The bottom right figure shows the result of the propagation of U_0 and the U_1^i sets: the purple ellipse is the image of the boundary of U_0 up to the time at which the splitting condition breaks, the blue asterisks are the images of the tracers, and the green small ellipsoids are the images of the boundaries of U_1^i.

To avoid this kind of behaviour of the tracers some new ones are added during the splitting procedure according to the following. If at the p-th splitting iterate the distance between two consecutive boundary tracers, $\{\sigma_p^1, \ldots, \sigma_p^m\}$, is greater than a certain tolerance, d_{tol}, this is $\|\sigma_p^i - \sigma_p^{i+1}\| > d_{tol}$, then a new (boundary) tracer $\tilde{\sigma}_p^{i+1}$ is added between them. To add a tracer we compute the mid point between σ_p^i and

σ_p^{i+1}, and normalise in order to set it on the boundary of U_0. In this way, the new tracer is

$$\tilde{\sigma}_p^{i+1} = x_0 + r_0 \frac{\sigma_p^i - \sigma_p^{i+1} - x_0}{\|\sigma_p^i - \sigma_p^{i+1} - x_0\|}.$$

Once a tracer is added at its corresponding place, the tracers with a super-index greater than it increase their index in one unit. The new tracer is propagated up to the current time in order to start the covering procedure.

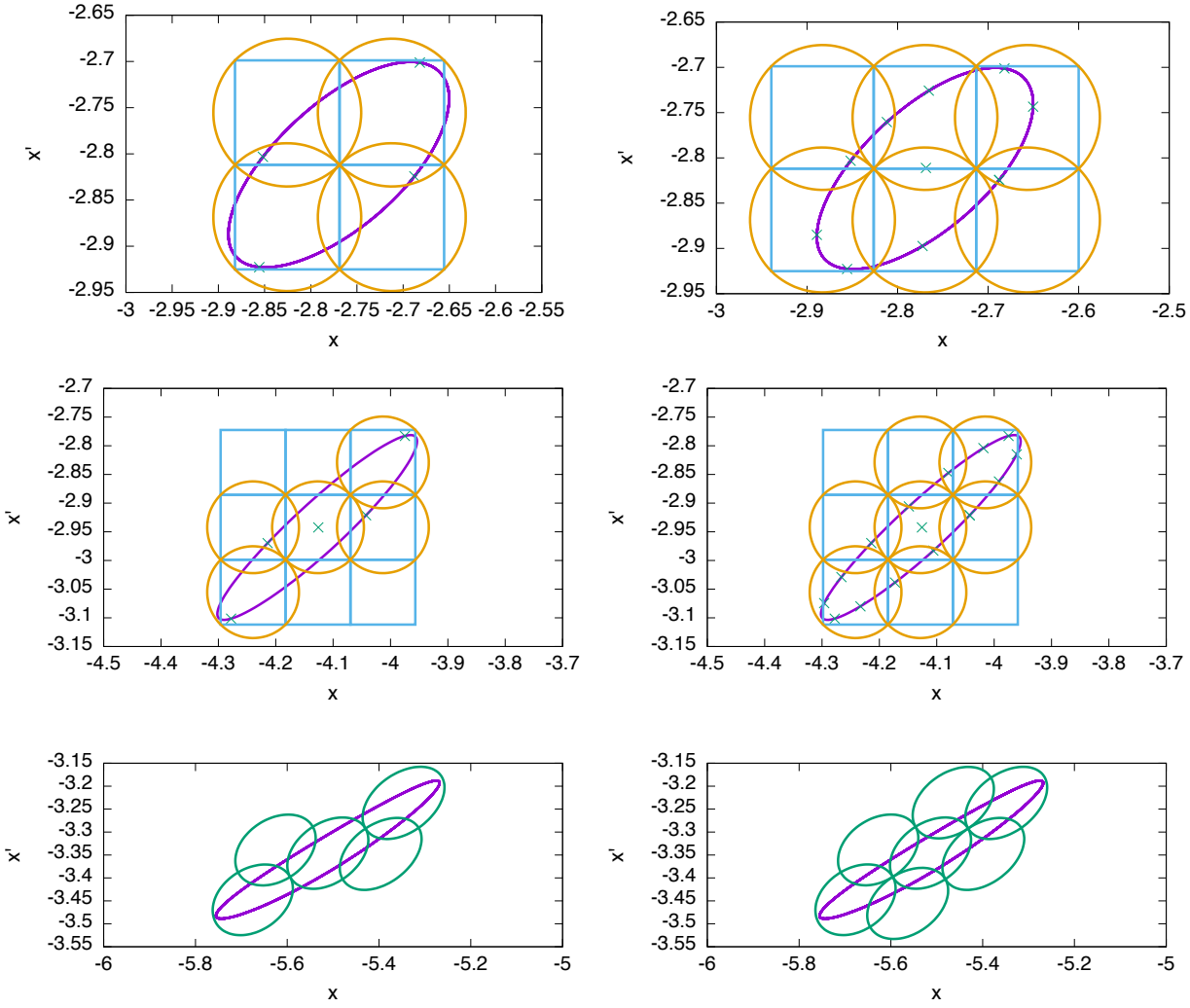

Figure 9: The first two left hand side plots show the results of the first and second covering iterates of Algorithm 1, and bottom left plot the result of the propagation up to the final time of U_0 and the U_1^i sets. The right hand side plots correspond to the same computations as in the left but adding tracers to the procedure when the initial ones are separated more than a certain distance $d_{tol} = 0.1$ (see more explanations in the text). The colour coding is the same as in Figure 8.

The three right hand side plots in Figure 9 show the second and third steps of the splitting procedure, together with the result at the final time, using the improved strategy of adding tracers with $d_{tol} = 0.1$.

Observe that now the final propagation of U_0 is well covered by the images of the U_p^i. However, the covering is not done with the minimum number of neighbourhoods. It is clear that the last step could be done using only four or five sets, instead of the seven required by the procedure. To overcome this drawback, it is convenient to align the grid along the maximum expansion direction. In this way there will be a direction with an elongated grid and narrower one in the orthogonal direction. This can be done using the following algorithm.

Algorithm 2 (Adapted direction to determine splittings using tracers)**.** *Once the propagation is stopped by the accuracy condition, then:*

1. *Find the most expanding direction v_e.*
2. *Use a Gram-Schmidt method to compute an orthonormal basis which contains $\vec{v}_e$ as one of the vectors. Let C be the matrix of the change of coordinates defined by this basis.*
3. *Transform the position of the tracers to the new base according to $\overline{\sigma}_p^i = C\sigma_p^i$.*
4. *Apply Algorithm 1 to determine the centres $\overline{c}_i$ of the neighbourhoods that will be propagated.*
5. *Transform the position of the centres back to the initial coordinate frame: $c_i = C^\top \overline{c}_i$.*

Note that the radii of the neighbourhoods does not change with this algorithm, since C is an orthonormal transformation.

Figure 10 displays the results obtained, for the same example and values of the parameters, using Algorithm 2. The bottom plot with the final results shows that, the number of neighbourhoods has been reduced from the seven needed by the unmodified Algorithm 1 to only four.

Observe that with Algorithm 2 the highly non-convex case is perfectly managed, since only such boxes in the grid containing a tracer will be covered by the new set of neighbourhoods, avoiding the overestimation produced by the previous method.

Algorithm 2 can be implemented in higher dimensions. The only thing that requires some attention is how to sort the tracers, and include additional ones when required. In the two-dimensional case, it is easy to sort them checking the distance between two consecutive ones. In the boundary of an n-dimensional sphere there is no such trivial sequence. The procedure starts with $2n+1$ tracers, $2n$ are located at each one of the intersections of the coordinate directions with the boundary of U_0, and an additional one is at the center of U_0.

If the dimension n is greater than 2, each tracer σ_0^i is identified by an index, as in the 2-dimensional case, but has also associated an array with the indices of the other tracers to which σ_0^i is connected. The length of the array is 2^{n-1}. Since the tracers also generate hyper-polyhedra, it is possible to add to the previous indices an array of all the polyhedra with the tracers that are at its vertices. Two tracers are in the same

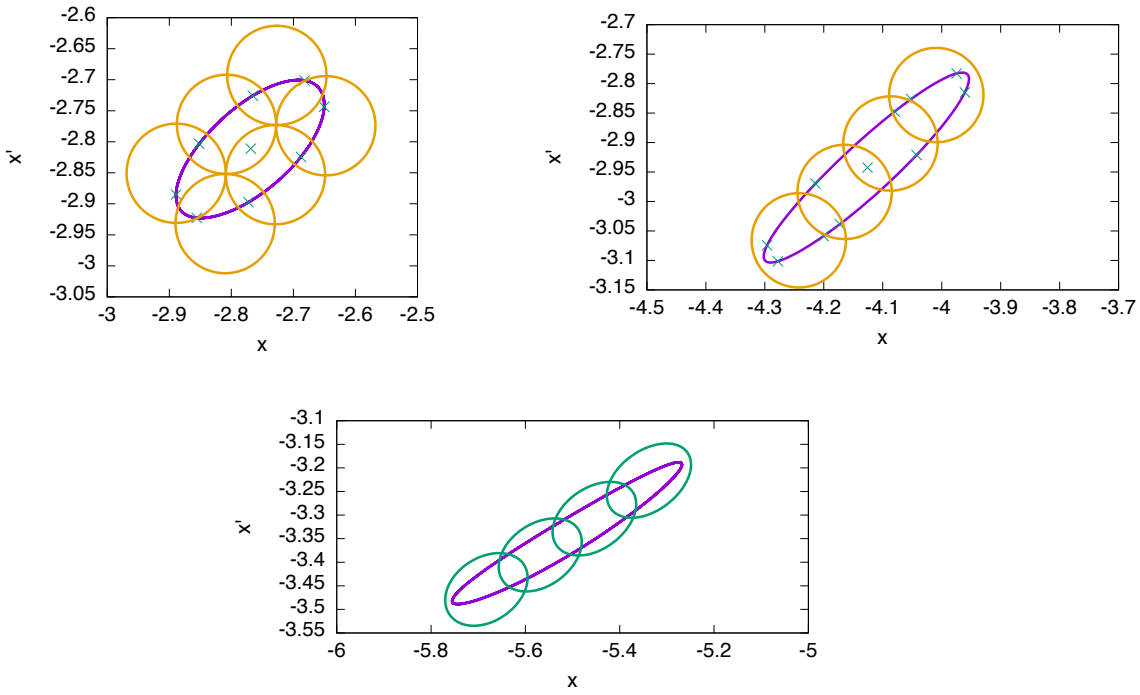

Figure 10: The two top plots show the results of the first and second splitting iterates using Algorithm 2, with the same values of the parameters used to get the results shown in Figure 9. The purple curve is the "exact" propagation of U_0, the green crosses are the propagation of the tracers and the yellow circles the sets U_p^i that are propagated. The bottom plot displays the result at the final integration time; the purple curve is the final neighbourhood and in green curves are the four U_p^i sets propagated in the last step.

polyhedron if they are connected by an edge. Therefore, the procedure stores the relation between two kinds of objects: a list of tracers and a list of polyhedra.

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

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

Observe that the volume criteria has a problem: it may happen that all the vertices of a polyhedron, except one, are close; in this situation the volume of the polyhedron can be small but there will be a tracer far away. In the 3D case the polyhedron will be a triangle. We can put a vertex of the triangle as far as we want and

have the other two in such a way that the surface (the 2-dimensional volume) be small. Therefore, there should be a new tracer in between. However, since the volume is small it is not added. Because of this, the volume-strategy has been discarded.

Recall that in the two-dimensional case when a tracer is added we normalise the barycenter of the two distant tracers. Following this idea, there are now two options for adding a tracer:

- Normalise the barycenter of the vertices of the polyhedron with distant tracers.
- Normalise the barycenter of the distant tracers, i.e: only the vertices of the polyhedron which are distant are shortened.

Figure 11 shows these two different ways of adding a new tracer. It can be seen that using the first option, adding the tracer at the barycenter of the polyhedron, the longest distance (in red) remains fixed and, therefore, the procedure will proceed (and in fact repeated indefinitely) without affecting the distance between the two tracers, whilst the second option reduces the distance between them; so the first option has been discarded.

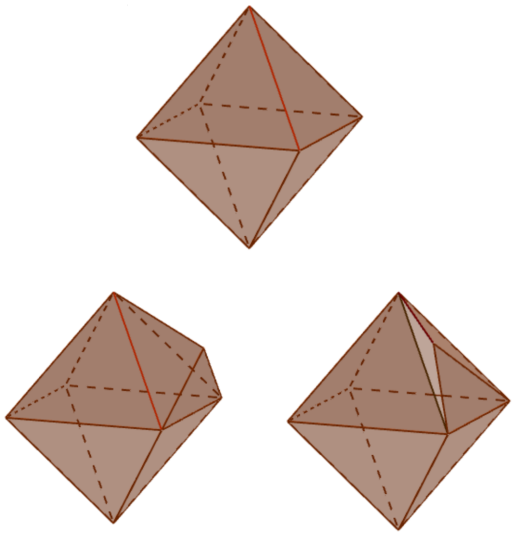

Figure 11: In the top figure we display an initial situation of the tracers (vertices of the polyhedron) in a 3-dimensional system. The red line between the two traces is assumed to be too long. The bottom figures show the two possible ways to add a new tracer. The left one shows how it is added using the face of the polyhedron in order to determine the new position: first the barycenter of all the vertices is computed and then normalised to be at a distance r_0. The right hand side one shows how to add the tracer using only the two tracers that are next to it. Again their barycenter is computed and then normalised to be at the right distance.

The output of the covering algorithm are $j+1$ sets of polynomials $\mathcal{P}_p,\ p = 0, \ldots, j$, where j is the number of divisions done. For each polynomial P_p^i in $\mathcal{P}_p$ we store the central point c_p^i and the coefficients of P_p^i. To

propagate an initial condition $\overline{x}_0 = x_0 + \xi$ from the initial time t_0 up to the final time T, $j+1$ polynomial evaluations are needed. At each step, p, the propagation of $\overline{x}_0$ up to the p-th division time is computed evaluating the polynomial of $\mathcal{P}_{p-1}$ whose central point is closer to $\overline{x}_{p-1}$, i.e.

$$\overline{x}_p = P^i_{p-1}(\overline{x}_{p-1} - c^i_{p-1}),$$

where i is the index such that $\|\overline{x}_{p-1} - c^i_{p-1}\|$ is minimum.

As concluding remark, we can say that the use of tracers allows to accurately cover U_0 using only a few number of U^i_p sets. The only parameters required by the procedure are the size r of the boxes of the grid, the tolerance d_{tol} used to add tracers, and the value ε_{JT} up to which we want to maintain the final precision. The smaller their values the better the approximations obtained. By reducing the size of the boxes of the grid more neighborhoods are propagated. By reducing the precision value ε_{JT} the number of splittings increases, since the split condition breaks more often.

Several tests have shown that, for most of the cases, the size of the initial neighborhood U_0 is a good starting value for r. A suitable value of the tolerance d_{tol}, that determines the distance at which a tracer must be added, is about one half of the size of the boxes of the grid.

5. Numerical Results

The two mentioned strategies, the ADS and the one considered in this paper, have been compared in two different dynamical systems:

- The simple pendulum, whose (normalised) differential equation of motion is:

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

  The dynamics of this system is well known. There are two equilibrium points at $x = 0$ and π and, in the phase space $(x, \dot{x})$, two regions with qualitatively different kinds of motion: one around the lower equilibrium position, which corresponds to the oscillations of the pendulum, and the other associated to the circulation regime (in fact there are two circulation regions, one with $\dot{x}(t) > 0$ and the other with $\dot{x}(t) < 0$). Between them there are two separatrices connecting the upper and bottom equilibria in an infinite time. It has been shown (see [14]) that the JT procedure is better suited for initial conditions not too close to the separatrices. We will show how the behavior of the JT is improved when the preceding subdivision algorithm is used.

- The two body problem, that studies the motion of two masses under their mutual gravitational attraction is studied. We consider the differential equation of relative motion of the bodies given by:

  $$\ddot{x} = -\mu \frac{x}{\|x\|^3}.$$

It is known that the orbits of this system are conic sections: ellipses, parabolas, hyperbolas and straight lines. We will only consider elliptic motions.

The comparison between the implementation of the two subdivision methods is not straightforward since they check the accuracy of the computations in different ways. In the ADS procedure the accuracy of the propagation is verified doing an estimation of the size of the $n+1$ order terms of the polynomial used for the propagation, based on an exponential fit of the size of all the known non-zero coefficients up to order n. In the approach presented in this paper the accuracy, that is estimated according to Section 2, is required to be below a given tolerance at each step of the method. Therefore, the tolerances that imply a certain subdivision are different. In order to have comparable results the tolerances have been tuned in such a way that the average of the errors using both methods be the same.

5.1. Propagation results for the pendulum

Two different computations have been done for the pendulum: one is the propagation of a regular point far from the separatrix, and the other is the propagation of a point on the separatrix.

5.1.1. The regular case

For this computation, a circulation orbit starting at $x_0 = (1, 0)$ has been propagated up to a final time of $t_f = 23$ adimensional time units. The maximum order of the polynomials used in the JT has been set equal to 3.

In order to check the correctness of the results obtained, they have been compared against two test propagations. The first one is a double-precision Runge-Kutta-Fehlberg-78 numerical integration (with a local truncation error equal to 10^{-14} that is assumed to provide a very accurate value for the flow; the errors obtained by the RKF-78 are lower than the errors obtained by a low degree polynomial approximation of order 3. The second propagation is the one obtained if no subdivision strategy is used, i.e. only a single polynomial propagation is used. The results obtained with this propagation show which are the gains of the developed procedure.

Figure 12 shows the errors of the three propagations. In all figures we display, using different colours, the logarithm of the differences between the result of the RKF-78 numerical integration and the polynomial approximation of the flow of degree 3 after 23 time units. In the left hand side column these errors are shown in the square box neighbourhood of initial conditions around the point x_0. The right hand side figures show the same errors around the image of x_0 after 23 time units.

As should be expected, if no divisions are done (first row) the results of the propagation are good around the central point $x_0 = (1, 0)$ and become worse when the distance to it increases. When the division algorithms are incorporated to the procedure, the precision is enhanced for points that are far away from the central point, as it is shown in the last two rows of the figure.

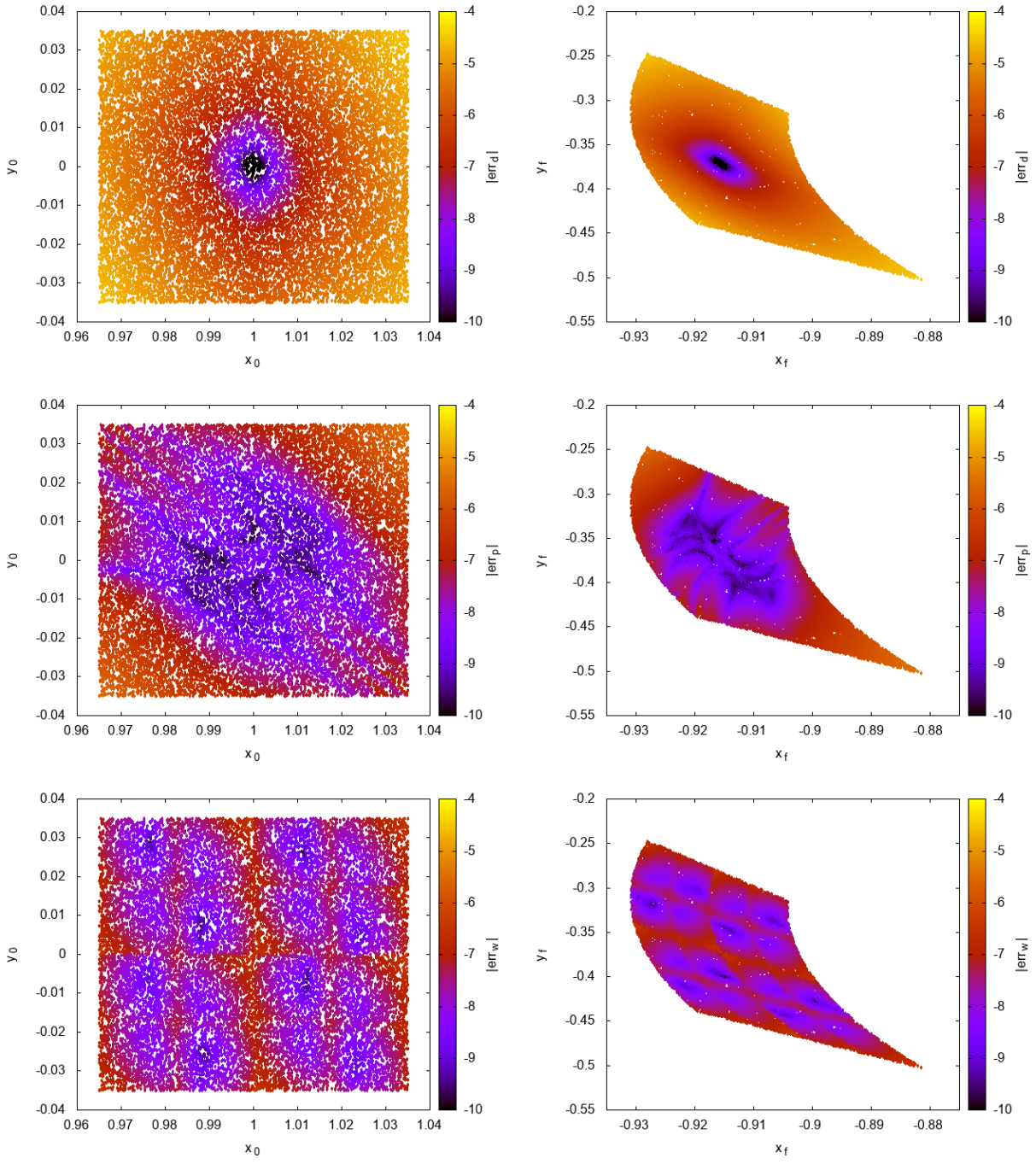

Figure 12: For the simple pendulum, errors obtained using the JT propagation without divisions (top), with the subdivision algorithm introduced in this paper (center), and using the ADS subdivision algorithm (bottom). In the left column the errors are displayed with respect to the initial state $(x_0, \dot{x}_0)$, and in the right column with respect to the final state $(x_f, \dot{x}_f)$. In the three cases, the initial condition is $(x_0, \dot{x}_0) = (1, 0)$, and a square box of 0.035 units is propagated during 23 time units.

The plots in the second row of Figure 12 show the accuracy obtained with the subdivision algorithm introduced in this paper. Clearly, the size of the set of points which are propagated with higher precision has increased by a factor lager than 2 when compared with the non-subdivision case. Note that, in these error plots, one cannot distinguish the subdivisions done by the procedure. This is due to the fact that at each subdivision step the new neighborhoods are set to cover in the best possible way the full domain, and the new neighborhoods to be propagated are selected independently from the previous polynomials.This is one of the major differences with respect to the ADS procedure, where at each division step a polynomial is split in two, and the new polynomials depend on the previous ones. The accuracy obtained with the ADS algorithm is shown in the last row of the figure. The division done by the method is detected by the eight "rectangles" corresponding to the regions with the lowest values of the errors. The central point of the grid is one of the points with highest error. This is because this point is, after the first splitting, at the region between two new polynomials and, therefore, far away from best propagated domains, that are those ones close to the central points associated to each step.

	No subdivisions	Subdivisions algorithm	ADS algorithm
log(error) average	-6.29	-8.07	-8.081
Maximum error	3.483941e-05	3.694106e-06	3.868810e-07
% of points worse than with no divisions	-	3.98%	9.13%
Number of polynomials stored	1	47	32
Total propagation time $\tau = \sum \tau_i$	23	81.6	385.28
CPU propagation time (s)	0.15	0.519	2.542
$2 \cdot 10^5$ evaluations CPU time (s)	0.107	0.667	0.131

Table 1: Summary of the results obtained in a pendulum JT propagation using and without using the subdivision algorithms.

Table 1 summarises some of the results displayed in Figure 12. For each of the three procedures, it gives the average of the logarithm of the errors obtained for the $2 \cdot 10^5$ samples used, as well as the maximum error. We can see that the two subdivision algorithms improve the final accuracy (average of the errors) by two orders of magnitude. The improvement in the maximum error is of one order of magnitude for the subdivisions algorithm and of two orders for the ADS algorithm.

For both methods, the percentage of points which have been propagated with less precision than the non-division propagation is also given in the above table. Although at first sight it may be surprising to have points with lower accuracy using the subdivision algorithm than without using it, this has a clear explanation: all those points are close to the initial condition and for them the non-subdivision propagation has the best accuracy. If no division is done, the central point is propagated up to the ODE numerical integrator algorithm accuracy, while in the splitting algorithms the central point is not propagated with such

high accuracy (but of course inside the threshold accuracy required).

The final number of polynomials stored in the memory is also given in Table 1. If no subdivision is made, there is only one polynomial stored, as it is obviously expected. The new covering algorithm stores more polynomials, since each time that a subdivision happens, a new set of polynomials is added, while in the ADS algorithm only one new additional polynomial must be added.

The propagation is done from an initial time t_0 up to a final time $t_f = 23$. However, each time that a division is required and new polynomials are added, the required time to propagate all the polynomials up to the final time is also added to the computation. For this reason, if we define τ_i as the time during which the polynomial P_i is propagated, this is: if $P_i(\xi) \approx \phi_{t_{f,i}}(t_{0,i}, x + \xi)$, then $\tau_i = t_{f,i} - t_{0,i}$. In this way, the total propagation time displayed in the table is given by $\tau = \sum \tau_i$.

As we can also see in the table, although the subdivision algorithm stores more polynomials than the ADS procedure, the total propagation time is shorter. The explanation for this fact is the following: in the subdivision algorithm, after the first subdivision four polynomials are needed; each time a subdivision point is reached the four polynomials are rearranged in order to cover the full region but, due to the dynamics of the pendulum along the trajectories considered, the algorithm does not need more than four polynomials at any subsequent step. Therefore, the final propagated time is close to four times the time required for a single polynomial propagation. However, all the polynomials must be stored in order to compute the full propagation. In the ADS procedure, each time that a subdivision is made it is not possible to combine it with a previous one and, consequently, the propagation time becomes larger.

The last two parameters displayed in Table 1 are related to the CPU time (with a 3GHz Intel Core i7) required to calculate the two main operations of the algorithms: the propagation of all the polynomials needed to warranty the precision together with the operations related to the subdivision procedure, and the propagation of the $2 \cdot 10^5$ sample points, once all the polynomials have been computed. The former mainly gives the computational cost of the propagation of the polynomials, and it is almost proportional to the total propagation time (the cost associated to the subdivisions is almost negligible). The second CPU time is the time required to evaluate the polynomials at set of samples around the initial condition x_0. The propagation of the $2 \cdot 10^5$ samples used to obtain the errors shown in Figure 12 using a RKF-78 propagation is of 47 s. We can see that the division algorithm does that task slower than a single JT propagation and introduces a small overhead due to the pre-computation of the correct polynomial used in the evaluation. The ADS algorithm requires almost no overhead, while the covering algorithm has an additional overhead due to the fact that multiple polynomial evaluations are required.

5.1.2. Non-regular point

The second test case consist in the propagation of a neighbourhood of a point $x_0 = (0, 2)$ of the separatrix. The time of propagation for this example is $t_f = 5$ adimensional time units, and the maximum order of the

polynomials used by the JT has been set equal to 5.

The same comparisons and indicators of the preceding example have been used. The figures obtained for the distribution of the errors with respect to the initial conditions using the three methods are similar to those ones of the previous case. However, due to the dynamics of the system near the separatrix, after some time, the propagation of the initial neighbourhood U_0 concentrates on a narrow band around the separatrix, therefore, such plots does not give information of the errors as a function of the final positions.

Table 2 summarises the results obtained in this case. One can see that both subdivision algorithms improve the final accuracy by two orders of magnitude in the average of the errors. Now, the improvement in the maximum error is of two orders for the subdivision algorithm, and of three orders for the ADS algorithm. This larger improvement is justified by the fact that it is not possible to approximate the flow map with enough accuracy using low order polynomials, therefore, either a higher order is used or more polynomials are introduced, as it is the case of the subdivision algorithms.

For the total number of polynomials stored, the ADS method requires less polynomials. Observe however that, since the division is done by two at some point of the propagation, the point that is propagated is no longer on the separatrix and, therefore, the propagation becomes more regular. This fact makes that the points near the separatrix are better propagated without using the subdivision algorithms than using them. On the other hand, the covering algorithm always covers the separatrix. Thus the propagation requires more polynomials but the number of samples that are better computed is higher.

	No subdivisions	Subdivisions algorithm	ADS algorithm
log(error) average	-5.23	-7.15	-7.07
Maximum error	4.60e-03	6.79e-05	4.65e-6
% of points worse than without divisions	-	0.660%	15.14%
Number of polynomials stored	1	14	10
Total propagation time $\tau = \sum \tau_i$	5	19	11.28
CPU propagation time (s)	0.101	0.371	0.274
$2 \cdot 10^5$ evaluations CPU time (s)	0.116	0.190	0.146

Table 2: Summary of the results obtained in a pendulum JT propagation around the separatrix with and without using the subdivision algorithms.

The total propagation times τ and the CPU times for the propagations and the sample evaluations shown does not show any relevant fact.

5.2. Propagation results for Kepler's problem

For Kepler's problem we have used as reference orbit one with adimensional initial condition $(x, y, \dot{x}, \dot{y}) = (1, 0, 0, \sqrt{1.5})$, that corresponds to an elliptical orbit around the Earth whose perigee is around 1000 km from its surface. We have used a square neighborhood around this initial condition of 0.035 adimensional units in each direction, that corresponds to approximately 200 km in positions and 276 m/s in velocities.

As in the case of the pendulum, the output results of a RKF-78 numerical integration has been taken as a reference, and we have used polynomial approximations of the flow of degree 5. Also a non-subdivision polynomial propagation has been used in order to see which are the benefits of using the subdivision procedures.

Figure 13 shows the accuracies of the three JT propagations. On the left hand side plots, the accuracy of each propagation is displayed against the initial distance to the central point x_0. One can observe that the non-subdivision propagation behaves as expected. The closer to the central point the better accuracy is achieved. However, that does not happen in the two subdivision algorithms. In both of them a bigger precision is obtained in average but the points closer to the central point are propagated with less accuracy than in the non-subdivision case. This happens because the methods change the independent term of the propagated polynomials each time that a subdivision is made and, therefore, the central point looses the high precision at which it is propagated in the non-division propagation. The right hand side plots of the figure show the logarithm of the error as a function of the final state (x_f, y_f). We can see that those points which are worse propagated are the ones that finish in an extreme position. Also it is worth to mention that, although the plots show the results in two dimensions, actually the space is four-dimensional. Therefore two points at the same configuration point may have different velocities.

Table 3 shows a summary of the results obtained. As in the previous cases, the division algorithms increase the accuracy of the results by one order in average. In this case, better accuracy is obtained due to two main factors: on one hand the integration time is smaller (in order to compensate the higher dimension), and on the other hand the order used in the JT is higher. A look to the maximum errors shows that the ADS algorithm has better accuracies than the covering algorithm, however the percentage of points which are worse propagated is higher.

The number of polynomials and the propagation time required by the ADS procedure is around twice the one needed by the covering method. As before, the relation between the CPU time and the total propagation time is almost linear. A small overhead from the expected linear behaviour (4 seconds in the covering algorithm and 3 seconds in the ADS algorithm) is detected due to the division procedure. The evaluation CPU time for the RK78 propagation is of 10.36 s. Again, the division algorithms are slower due to the fact that the appropriate polynomial must be located to do the evaluation. The covering algorithm also requires additional evaluation time due to the several polynomial evaluations that must be done.

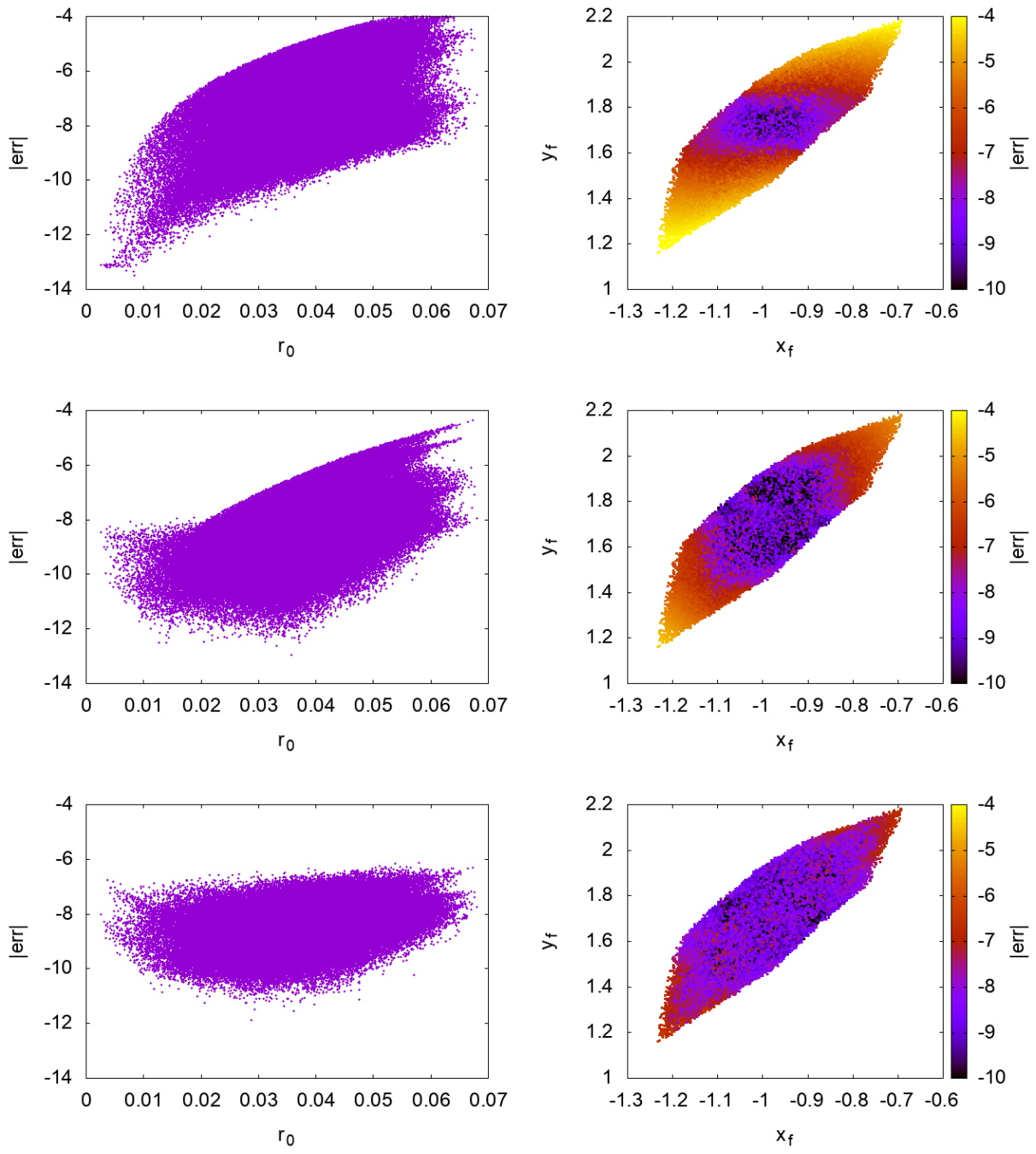

Figure 13: For the Kepler problem, errors obtained using the JT propagation without divisions (top), with the subdivision algorithm introduced in this paper (center), and using the ADS subdivision algorithm (bottom). In the left column the decimal logarithm of the errors are displayed with respect to the initial distance to the center. In the right column the decimal logarithm error is plotted in colour with respect to the final state (x_f, y_f) (without taking into account the velocity of the point). In all cases the initial condition is the point $(x_0, y_0, \dot{x}_0, \dot{y}_0) = (1, 0, \sqrt{1.5}, 0)$ in a square box of 0.035 non-dimensional units propagated during 3 time units.

	No subdivisions	Subdivisions algorithm	ADS algorithm
log(error) average	-7.77828	-8.99259	-8.97618,
Maximum error	1.393011e-04	3.564957e-05	5.803936e-07
% of points worse than without subdivisions	-	17.7086%	22.5463%
Number of polynomials stored	1	35	78
Total propagation time $\tau = \sum \tau_i$	3	47.40	99.44
CPU propagation time (s)	3.36	57.43	114.34
$2 \cdot 10^5$ evaluations CPU time (s)	0.5	1.42	0.57

Table 3: Summary of the results obtained in a Kepler's problem propagation using and without using division algorithms.

6. Conclusions

A new subdivision algorithm has been introduced to improve the numerical propagation of ODE using Jet Transport procedures. The algorithm has been compared against the Automatic Domain Splitting method, as well as the non-subdivision basic strategy.

The developed procedure uses the more expansive and contractive directions of the flow to locate the covering sets. When needed, the number of sets, together with the associated polynomials, either increases adding new sets or decreases merging some of them, always keeping the required accuracy in the computations.

The developed algorithm looses the property, characteristic of the Jet Transport propagations, of keeping the expansion of the flow up to an arbitrary order at the final time. In the new algorithm, several polynomials are stored at each subdivision step and its composition provides the final value of the propagation. The polynomial composition must be done independently for each sample. Unfortunately, due to both the nature of the algorithm and to the final function to be approximated, this drawback cannot be avoided.

An important characteristic of the Jet Transport propagation using Taylor expansions is its good local accuracy properties around the central point of propagation. Other expansions using Chebyshev or Hermite polynomials, can be useful to propagate uncertainties without the dependency of the central propagation point or with a wider convergence radius but, for a given accuracy, it is not clear which one is better from the computational point of view.

Acknowledgements

The authors are very grateful to the anonymous reviewers for their helpful comments and suggestions on revising the paper. This work has been partially supported by the grants MTM2013-41168-P, AP2010-0268 (D.P.), MTM2013-41168-P, MTM2016-80117-P (G.G.), and MTM2012-31714, 2014SGR504 (J.J.M.).

References

[1] A. Abad, R. Barrio, F. Blesa, and M. Rodríguez. *Algorithm924: TIDES, a Taylor Series Integrator for Differential Equations. ACM Trans. Math. Softw.*, 39(1):1–28, 2012.

[2] E.M. Alessi, C. Simó, A. Farrès, A. Vieiro, and À. Jorba. *Efficient Usage of Self Validated Integrators for Space Applications.* Ariadna Final Report, Apr 2008.

[3] E.M. Alessi, A. Farrès, A. Vieiro, À. Jorba, and C. Simó. *Jet Transport and Applications to NEO's.* Proceedings of 1st IAA Planetary Defense Conference. ESA Conference Bureau, Granada (Spain), 2009.

[4] R. Armellin, P. Di Lizia, F. Bernelli-Zazzera, and M. Berz. *Asteroid Close Encounters Characterization using Differential Algebra: The case of Apophis.* Celestial Mechanics and Dynamical Astronomy, 107(4), 2010, 451-470.

[5] R. Armellin, and P. Di Lizia. *Probabilistic Optical and Radar Initial Orbit Determination.* Journal of Guidance, Control and Dynamics, http://dx.doi.org/10.2514/1.G002217

[6] F. Bernelli-Zazzera, P. Di Lizia, M. Vasile, and M. Massari. *Assessing the Accuracy of Interval Arithmetic Estimates in Space Flight Mechanics.* Ariadna Final Report, 2005.

[7] M. Berz. *Modern Map Methods in Particle Beam Physics.* Academic Press, New York, 1999.

[8] À. Jorba. *A Methodology for the Numerical Computation of Normal Forms, Centre Manifolds and First Integrals of Hamiltonian Systems.* Exp. Math. 8(2):155-195, 1999.

[9] À. Jorba, and M. Zou. *A Software Package for the Numerical Integration of ODE's by Means of High-Order Taylor Methods.* Exp. Math., 14(1):99-117, 2005.

[10] K. Makino, and M. Berz. *COSY INFINITY 9.1 Programmer's Manual. MSU Report MSUHEP101214*, Michigan State University, East Lansing, MI., 2011.

[11] PARI/GP, version `2.5.3`. Available at: http://pari.math.u-bordeaux.fr/.

[12] D. Pérez-Palau, G. Gómez, and J.J. Masdemont. *Detecting Invariant Manifolds Using Hyperbolic Lagrangian Coherent Structures.* Paper number IAA-AAS-DyCoSS1-08-06, IAA Conference on Dynamics and Control of Space Systems. Porto (Portugal), 2012.

[13] D. Pérez-Palau. *Dynamical Transport Mechanisms in Celestial Mechanics and Astrodynamics Problems.* PhD dissertation. Universitat de Barcelona, 2015.

[14] D. Pérez-Palau, J.J. Masdemont, and G. Gómez. *Tools to Detect Structures in Dynamical Systems Using Jet Transport.* Celestial Mechanics and Dynamical Astronomy, 123(3)239–262, 2015.

[15] M. Rasotto, A. Morselli, A. Wittig, M. Massari, P. Di Lizia, R. Armellin, C. Valles, and G. Ortega. *Differential Algebra Space Toolbox for Nonlinear Uncertainty Propagation in Space Dynamics.* The 6th International Conference on Astrodynamics Tools and Techniques (ICATT), Darmstadt (Germany), 2016.

[16] C. Simó. *Global Dynamics and Fast Indicators.* Global Analysis of Dynamical Systems, 373-389, Inst. Phys., Bristol (United Kingdom), 2001.

[17] V. Szebehely. *Theory of Orbits: The Restricted Problem of Three Bodies.* Academic Press, 1967.

[18] M. Valli, R. Armellin, P. Di Lizia, and M. Lavagna. *Nonlinear Filtering Methods for Spacecraft Navigation Based on Differential Algebra.* Acta Astronautica, 94(1)363-374, 2014.

[19] D. Wilczak. *CAPD DynSys Library Documentation. The CAPD group.* Available at: mnich.ii.uj.edu.pl/capd/doc4.

[20] A. Wittig, P. Di Lizia, R. Armellin, K. Makino, F. Bernelli-Zazzera, and M. Berz. *Propagation of Large Uncertainty Sets in Orbital Dynamics by Automatic Domain Splitting.* Celestial Mechanics and Dynamical Astronomy, 122(3)239–261, 2015.

[21] A. Wittig, C. Colombo, and R. Armellin. *Long-Term Density Evolution through Semi-Analytical and Differential Algebra Techniques.* Celestial Mechanics and Dynamical Astronomy, 2017, http://link.springer.com/article/10.1007/s10569-017-9756-x