Monografías de la Real Academia de Ciencias de Zaragoza **35**, 83–95, (2011).

TIDES: A free software based on the Taylor series method

M. Rodríguez[1,2], A. Abad[1], R. Barrio[1] and F. Blesa[3]

[1] GME and IUMA, Universidad de Zaragoza, 50009 Zaragoza, Spain

[2] Centro Universitario de la Defensa de Zaragoza, 50090 Zaragoza, Spain

[3] GME and Depto. Física Aplicada, Universidad de Zaragoza, 50009 Zaragoza, Spain

Abstract

In many branches of the science, the numerical solution of differential equations is a natural request. However, the way to solve those equations is not always the same. Sometimes it is needed to solve the equation as fast as possible. In other cases good precision is required, even 100 digits or more to guarantee the final results. Sensitivity analysis also may be wanted in some integrators. In this paper, a new integration software based on the Taylor series method is introduced, with all these features. It can provide fast integrations that compete with well established codes and it is ready for high precision integrations with as many digits as desired. Besides it can compute the partial derivatives of the variables with respect to initial conditions or parameters of the system up to any order. This software has been written focusing in the philosophy *easy-to-use*, so with very simple commands the necessary code is generated automatically. At the end, we also present some numerical results to illustrate its competitiveness and high performance with chaotic problems.

1 INTRODUCTION

The Taylor Series Method (TSM) is an integration method with a long history. Classical authors like Newton or Euler already used power series expansions as integration methods. However, nowadays it has been rediscovered with the implementation of automatic differentiation [11] techniques which allow us to transform a theoretical method into a practical one. Afterwards, it has reached a good position as an integration method and many research groups [5, 6, 15, 19, 20] have adopted this method as their main integrator algorithm.

This method is widely used in the study of Dynamical Systems because, apart from being a competitive integration method, it is very flexible, that is, it can be easily adapted

for many purposes like the study of partial derivatives [8], high precision calculations [1, 9, 15] or detection of events. It also may be implemented by using interval arithmetic in order to attain a rigorous method of integration [10,17,23] which guarantees the results, a needful step in Computer Aided Proofs [13, 16].

Here we present an implementation of the Taylor Series Method adapted to integrate ODEs, partial derivatives up to any order and ready to use multiple precision if needed. A piece of the software has been written in MATHEMATICA under the *easy-to-use* philosophy, so with very simple commands we will have all the necessary code for our purposes.

The main idea of the Taylor Method is very simple. Let is consider the initial value problem:

$$\frac{d\boldsymbol{y}(t)}{dt} = \boldsymbol{f}(t, \boldsymbol{y}(t); \boldsymbol{p}), \qquad \boldsymbol{y}(t_0) = \boldsymbol{y}_0, \qquad t \in \mathbb{R},\, \boldsymbol{y} \in \mathbb{R}^s,\, \boldsymbol{p} \in \mathbb{R}^k. \tag{1}$$

where $\boldsymbol{p}$ represents a vector of $k \geq 0$ parameters.

Now, the value of the solution at $t_{i+1} = t_i + h_{i+1}$ (that is, $\boldsymbol{y}(t_{i+1})$) is approximated from the n-th degree Taylor series of $\boldsymbol{y}(t)$ developed at t_i and evaluated at $t = t_{i+1}$ (the function $\boldsymbol{f}$ has to be a smooth function, in this paper we will consider $\boldsymbol{f}$ to be analytic).

$$\begin{aligned}
\boldsymbol{y}(t_0) &\stackrel{\text{def}}{=} \boldsymbol{y}_0, \\
\boldsymbol{y}(t_{i+1}) &\simeq \boldsymbol{y}(t_i) + \frac{d\boldsymbol{y}(t_i)}{dt}\, h_{i+1} + \frac{1}{2!}\, \frac{d^2\boldsymbol{y}(t_i)}{dt^2}\, h_{i+1}^2 + \ldots + \frac{1}{n!}\, \frac{d^n\boldsymbol{y}(t_i)}{dt^n}\, h_{i+1}^n \\
&\simeq \boldsymbol{y}_i + \boldsymbol{f}(t_i, \boldsymbol{y}_i)\, h_{i+1} + \frac{1}{2!}\, \frac{d\boldsymbol{f}(t_i, \boldsymbol{y}_i)}{dt}\, h_{i+1}^2 + \ldots + \frac{1}{n!}\, \frac{d^{n-1}\boldsymbol{f}(t_i, \boldsymbol{y}_i)}{dt^{n-1}}\, h_{i+1}^n \stackrel{\text{def}}{=} \boldsymbol{y}_{i+1}.
\end{aligned} \tag{2}$$

The practical way of computing the coefficients $\{d^j\boldsymbol{y}(t_j)/dt^j\}_j$ is using the automatic differentiation techniques [12,21]. In the literature we can find many software implementations of the TSM which compute these coefficients automatically, for example ATOMFT (Corliss et al.), COSY INFINITY (Berz et al.), DAETS (Nedialkov et al.), TAYLOR (Jorba and Zou) and, of course, TIDES.

2 AUTOMATIC DIFFERENTIATION WITH PARTIAL DERIVATIVES

The practical calculation of the coefficients of a Taylor series expansion of a function is done with the classical rules of automatic differentiation. Automatic differentiation gives us a recursive procedure to obtain the numerical value of reiterated derivatives with no error (but the round off one).

The technique is based on the decomposition of the function into a sequence of arithmetic operations and calls to standard unary functions. For example, if we consider $f(x) = \log(x^2 + 1)/(x^2 - 1) + x$, we evaluate the function starting from the value of the

variable x and obtain the value of $f(x)$ as follows:

$$\begin{aligned} l_1 &= x, \\ l_2 &= l_1\, l_1, \\ l_3 &= l_2 + 1, \\ l_4 &= l_2 - 1, \\ l_5 &= \log l_3, \\ l_6 &= l_5/l_4, \\ l_7 &= l_6 + l_1 \\ f(x) &= l_7. \end{aligned} \tag{3}$$

and by applying differentiation rules at each step we can obtain the value of the derivatives of f for any order. Once we have these values, from equation (1) we obtain the coefficients of the Taylor series of the solution y:

$$\boldsymbol{y}^{[j]}(t) \stackrel{\text{def}}{=} \frac{1}{j!}\frac{d^j \boldsymbol{y}(t)}{dt^j} = \frac{1}{j!}\frac{d^{j-1}\boldsymbol{f}(t)}{dt^{j-1}} = \frac{1}{j}\boldsymbol{f}^{[j-1]} \tag{4}$$

If we also want to calculate the Taylor coefficients of the partial derivatives of the solution with respect to some parameter or initial condition we can use also automatic differentiation. First we introduce some notation. We will call $\mathbb{N}_0$ the set of the natural numbers with zero, $\mathbf{i} = (i_1, \ldots, i_s) \in \mathbb{N}_0^s$ a multi-index with the partial derivative orders, $\mathbf{i}^* = \mathbf{i} - (0, \ldots, 0, 1, 0, \ldots, 0)$, that is $\mathbf{i}^* = (0, \ldots, 0, i_k - 1, i_{k+1}, \ldots, i_s)$ (we subtract 1 to de first non-zero element of $\mathbf{i}$), $|\mathbf{i}| = \sum i_j$ the total order of partial derivation, and $\mathbf{i} - \mathbf{j} = (i_1 - j_1, \ldots, i_s - j_s)$. For $\mathbf{v} \in \mathbb{N}_0^s$, we define the multi-combinatorial number $\binom{\mathbf{i}}{\mathbf{v}} = \binom{i_1}{v_1} \cdots \binom{i_s}{i_s}$. Finally we define the the Taylor j-th coefficient of the partial derivative of $\boldsymbol{f}$ as:

$$\boldsymbol{f}_{\mathbf{i}}^{[j]}(t) = \frac{\partial^{i_1+\ldots+i_s}\boldsymbol{f}^{[j]}(t)}{\partial y_1^{i_1} \cdots \partial y_s^{i_s}} \tag{5}$$

and

$$\tilde{h}_{\mathbf{v},\mathbf{i}}^{[j,n]} = h_{\mathbf{v}}^{[j]} \text{ if } (j \neq n \text{ or } \mathbf{v} \neq \mathbf{i}); \quad \tilde{h}_{\mathbf{v},\mathbf{v}}^{[n,n]} = 0. \tag{6}$$

The basic rules of automatic differentiation extended with partial derivatives can be found in [8] (with the same notation). We show as an example the rule of the logarithm.

Proposition 1. *Let f be a function of class $\mathcal{C}^n$. Then, if we define $h(t) = \log f(t)$, the automatic differentiation rule for h is:*

$$\begin{aligned} h_{\mathbf{0}}^{[0]} &= \log\left(f^{[0]}(t)\right), \\ h_{\mathbf{i}}^{[0]} &= \frac{1}{f^{[0]}}\left(f_{\mathbf{i}}^{[0]} - \sum_{\mathbf{0}<\mathbf{v}\leq\mathbf{i}^*}\binom{\mathbf{i}^*}{\mathbf{v}} h_{\mathbf{i}-\mathbf{v}}^{[0]} \cdot f_{\mathbf{v}}^{[0]}\right), \qquad \mathbf{i} > \mathbf{0}, \\ h_{\mathbf{i}}^{[n]} &= \frac{1}{f^{[0]}}\left(f_{\mathbf{i}}^{[n]} - \frac{1}{n}\sum_{j=0}^{n-1}(n-j)\left\{\sum_{\mathbf{v}\leq\mathbf{i}}\binom{\mathbf{i}}{\mathbf{v}}\tilde{h}_{\mathbf{i}-\mathbf{v},\mathbf{i}}^{[n-j,n]} \cdot f_{\mathbf{v}}^{[j]}\right\}\right), \quad n > 0. \end{aligned} \tag{7}$$

Proof. For $h^0_{\mathbf{i}}$, first we derive with respect $\mathbf{e_k}$ (k is the first non-zero component of $\mathbf{i}$) and we have the relation $f^{[0]}_{\mathbf{e_k}} = h^{[0]}_{\mathbf{e_k}} \cdot f$. Now we perform $\mathbf{i} - \mathbf{e_k} = \mathbf{i}^*$ derivatives by applying the Leibniz rule for partial derivation of the product (see [8]) and we have $f^{[0]}_{\mathbf{i}} = \sum_{\mathbf{v}\leq\mathbf{i}^*} \binom{\mathbf{i}^*}{\mathbf{v}} h^{[0]}_{\mathbf{i}-\mathbf{v}} \cdot f^{[0]}_{\mathbf{v}}$. Finally we just have to work out the value of $h^{[0]}_{\mathbf{i}}$.

For $h^{[n]}_{\mathbf{i}}$, we first derive once with respect to time to reach the expression $f^{[1]} = h^{[1]} \cdot f$. Then we apply the well known automatic differentiation rule of the product [21] to reach

$$nf^{[n]} = \sum_{j=0}^{n-1} (n-j) h^{[n-j]} \cdot f^{[j]}$$

. At this point we derive with respect to $\mathbf{i}$ applying the Leibniz rule again:

$$nf^{[n]}_{\mathbf{i}} = \sum_{j=0}^{n-j} (n-j) \sum_{\mathbf{v}\leq\mathbf{i}} \binom{\mathbf{i}}{\mathbf{v}} h^{[n-j]}_{\mathbf{i}-\mathbf{v}} \cdot f^{[j]}_{\mathbf{v}}$$

. Finally, considering the definition of $\tilde{h}$ (6), we work out the value of $h^{[n]}_{\mathbf{i}}$. □

2.1 Dealing with partial derivatives

Here we are going to explain briefly how MATHEMATICA deals with the partial derivatives and how it transforms expressions of the form $\mathbf{v} \leq \mathbf{i}$ into sums with natural indexes which can be traduced into C or Fortran codes. We call $\mathcal{D}^m_s = \{\mathbf{i}, |\mathbf{i}| \leq m\}$, the set of all partial derivatives of the function f of order less or equal to m. We may easily define a total order relationship in this set by considering $\mathbf{i} < \mathbf{j}$ if $\sum i_k < \sum j_k$ or $\sum i_k = \sum j_k$ and there exists an index p such as $i_k = j_k$, $\forall k < p$, and $i_p < j_p$, and $\mathbf{i} = \mathbf{j}$ if $i_k = j_k$, $\forall k$.

This order relationship is the one used in the extended formulas of automatic differentiation (see equation (7) for example). MATHEMATICA, for each index $\mathbf{i}$ computes the set $\mathcal{V}(\mathbf{i}) = \{\mathbf{v}, \mathbf{v} \leq \mathbf{i}\}$, the set of all partial derivatives needed to perform the calculation of the $\mathbf{i}$ partial derivative. If instead of one partial derivative we have a set of partial derivatives $\mathcal{D}$, we define $\mathcal{V}(\mathcal{D}) = \bigcup_{\mathbf{i}\in\mathcal{D}} \mathcal{V}(\mathbf{i}) \subset \mathcal{D}^m_s$. Then we order this set and identify each element of the set $\mathcal{V}(\mathcal{D})$ with an integer number between 0 and $N-1$, with $N = \sharp\mathcal{D}$ the number of elements of $\mathcal{V}(\mathcal{D})$. This index will represent the position in the ordered set $\mathcal{V}(\mathcal{D})$.

We notice that in the process of computing partial derivatives in the Taylor series method only two kind of expressions are involved (see equation (7)):

$$\sum_{\mathbf{v},\mathbf{w}\in\mathcal{V}(\mathbf{i})} C_{\mathbf{i},\mathbf{v}} f^{[-]}_{\mathbf{v}} g^{[-]}_{\mathbf{w}}, \qquad \sum_{\mathbf{v},\mathbf{w}\in\mathcal{V}(\mathbf{i}^*)} C_{\mathbf{i}^*,\mathbf{v}} f^{[-]}_{\mathbf{v}} g^{[-]}_{\mathbf{w}}, \quad \mathbf{w} = \mathbf{i} - \mathbf{v}, \quad C_{\mathbf{i},\mathbf{v}} = \binom{\mathbf{i}}{\mathbf{v}},$$

where $\mathbf{i}$ goes over the set $\mathcal{V}(\mathcal{D})$.

In order to write the code to compute the Taylor series of the partial derivatives of the solution of an ODE we need to know the sets $\mathcal{V}(\mathbf{i}), \mathcal{V}(\mathbf{i}^*)$ of every derivative that appears

in the process. The MATHEMATICA package MathTIDES computes automatically these sets and uses them to write the final C code. The way to perform this is by building two lists of integers that represent the indexes for $\mathbf{v}$, and another two lists for the indexes of $\mathbf{w} = \mathbf{i} - \mathbf{v}$. MATHEMATICA also computes the lists with the integer value of the multi-combinatorial numbers $C_{\mathbf{i},\mathbf{v}}$ and $C_{\mathbf{i}^*,\mathbf{v}}$. Finally, MATHEMATICA also writes two lists with the initial and final position in each list for each value of $\mathbf{i}$ and $\mathbf{i}^*$.

We show the process with an example. If we want to compute the partial derivative

$$\frac{\partial^4 f}{\partial y_1^3 \partial y_2}$$

first we construct the set of all derivatives we want to compute: $\mathcal{D} = \{(3,1)\}$. Now we need to calculate the set of all necessary derivatives for the elements of $\mathcal{D}$, to order that set, and to identify each element with an integer:

$$\mathcal{V}(\mathcal{D}) = \{(0,0),(1,0),(0,1),(2,0),(1,1),(3,0),(2,1),(3,1)\} = \{0,1,2,3,4,5,6,7\}$$

And for each element we will have:

$$\begin{aligned}
\mathcal{V}(0) &= \mathcal{V}((0,0)) &&= \{(0,0)\} = \{0\} \\
\mathcal{V}(1) &= \mathcal{V}((1,0)) &&= \{(0,0),(1,0)\} = \{0,1\} \\
\mathcal{V}(2) &= \mathcal{V}((0,1)) &&= \{(0,0),(0,1)\} = \{0,2\} \\
\mathcal{V}(3) &= \mathcal{V}((2,0)) &&= \{(0,0),(1,0),(2,0)\} = \{0,1,3\} \\
\mathcal{V}(4) &= \mathcal{V}((1,1)) &&= \{(0,0),(1,0),(0,1),(1,1)\} = \{0,1,2,4\} \\
\mathcal{V}(5) &= \mathcal{V}((3,0)) &&= \{(0,0),(1,0),(2,0),(3,0)\} = \{0,1,3,5\} \\
\mathcal{V}(6) &= \mathcal{V}((2,1)) &&= \{(0,0),(1,0),(0,1),(2,0),(1,1),(2,1)\} = \{0,1,2,3,4,6\} \\
\mathcal{V}(7) &= \mathcal{V}((3,1)) &&= \{(0,0),(1,0),(0,1),(2,0),(1,1),(3,0),(2,1),(3,1)\} = \{0,1,2,3,4,5,6,7\}
\end{aligned}$$

3 MULTIPLE PRECISION LIBRARIES

In this section we are going to comment briefly the high precision libraries used and the reason for choosing them. In the scientific community there are many libraries which implement arbitrary precision. Different libraries have different ways to represent arbitrary-precision numbers so the election of one or other library will determine the program. There are libraries for C (GMP, MPFR, IMSL, decNUMBER), Fortran (FMLIB, ARPREC, MPFUN), java (APFLOAT), C++ (APFLOAT), etc. Also these libraries are designed to manage only integer numbers (LiDIA), real numbers (almost all of them) or even complex arithmetic (MPFR).

Another important aspect when choosing one of them is the kind of license it has. We have avoided commercial or proprietary licenses in order to be able to distribute it within our software, so good options are GPL (GNU Public License) and LGPL (Lesser GNU Public License).

Finally we wanted an *easy-to-use* library which fits properly in the global philosophy of TIDES. The library having all these features is MPFR[1]. To compile and use this library it is necessary to have compiled and installed the high precision library of the GNU project GMP[2] both with LGPL license.

The use of the libraries is internal and the program can be used easily with the interface in MATHEMATICA as it is described in the next section.

4 USE OF TIDES

In this section we are going to explain in the first place the structure of TIDES. Afterwards we will comment the basic commands in MATHEMATICA necessary to use the library.

4.1 Structure of TIDES

TIDES has two different parts (pieces of software): The MATHEMATICA package MathTIDES and the C library libTIDES. The structure is shown in this table:

TIDES		
Product		Language
MathTIDES	preprocessor	MATHEMATICA
libTIDES	library (objects or source code)	C

MATHEMATICA builds automatically the necessary code to integrate the differential equation. The files generated by MATHEMATICA usually are enough to solve the equation, therefore the user just has to compile and execute them. If the user wants additional features, such as high precision or partial derivatives of the solution, the use of the library libTIDES is required.

Depending on the demands of the user one engine or other engine is used. This table illustrates the features of the different engines of TIDES:

[1] `http://www.mpfr.org`
[2] `http://gmplib.org`

Engine	Contents	mathTIDES generates	linked with
minf-tides	basic TSM	Fortran files	
minc-tides	basic TSM	C files	
dp-tides	complete TSM with partial derivatives	C files	libTIDES
mp-tides	complete TSM with partial derivatives with arbitrary precision	C files	libTIDES GMP library MPFR library

The two first engines (minf-tides and minc-tides) are minimal versions thought to be fast and simple. They are developed to integrate ordinary differential equations in which only simple functions appear: $x \pm y$, $x \times y$, x/y, $1/x$, $x^\alpha\,(\alpha \in \mathbb{R})$, $\alpha^x\,(\alpha > 0)$, e^x, $\log x$, $\sin x$ and $\cos x$. In these cases MATHEMATICA generates files (C or Fortran) with the integration engine, the implementation of the automatic differentiation (without partial derivatives) and the driver (main program).

If we want to integrate hyperbolic functions or inverse of trigonometric or inverse of hyperbolic functions, we have to use the next engine (dp-tides). It is also necessary to use this engine if we want to integrate with partial derivatives. MathTIDES generates files with the implementation of the automatic differentiation for the function, and with the driver (main program). In this case, the engine is included in the library libTIDES, so it is needed to link with this library when compiling.

The last engine, mp-tides, is the complete integrator with all the features: all the functions, partial derivatives and arbitrary precision. MathTIDES builds, like in the case of dp-tides, files with automatic differentiation and with the driver (main program). The engine is also in library libTIDES, but when compiling it is also required to link with high precision libraries GMP and MPFR (GMP needed by MPFR).

4.2 The MATHEMATICA *interface*

Along this subsection we are going to see the easy-to-use concept underlying in TIDES. We are going to explain the basic commands taking the example of the Lorenz equations [18]:

$$\dot{x} = -\sigma(x+y), \quad \dot{y} = -xz + rx - y, \quad \dot{z} = xy - bz \tag{8}$$

First we load the package with the command `<<MathTIDES`` , and then we can use the basic commands: `FirstOrderODE` and `CodeFiles`. The first one generates the internal expression that represents the differential equation. The second one generates the codes. To generate the differential equation other commands can be used, `NthOrderODE`, `PotentialToODE` or `HamiltonianToODE`, which creates the differential system from a n-th

order differential equation, a potential or a Hamiltonian, respectively. In our case we have to use `FirstOrderODE`. The arguments are: a list with the symbolic expression of the ODE, the independent variable, a list with the dependent variables and optionally a list with the parameters that appear in the equation. So, the command is:

```
lorenz = FirstOrderODE[{s(y-x), -x z + r x - y, x y - b z},
              t, {x, y, z}, {s, r, b}]
```

Once we have created the differential equation, we just have to execute the command `CodeFiles` according to our needs. The first argument is the equation, in our case `lorenz`. The second one is an string with the desired name for the files in C or Fortran. The next arguments are optional. These are the basic ones:

- `MinTIDES->"Fortran"` or `MinTIDES->"C"`. This generates the appropriated files to use the minimal engines in Fortran or C.
- `Output->Screen`, `Output->"filename"` or `Output->False`. Specifies the output stream, on the screen or in a file called `filename`. If `False` (default value) no output is shown or written (the final result is in an output variable).
- `PrecisionDigits->n`. With this option we control the number of digits of precision we are working with. If greater than 16 the engine mp-tides and all its libraries are required. The default value is 16.
- `AddPartials->{{x, y}, n, Until}` or `AddPartials->{{x, s}, n, Only}`. With this option we add the calculation of partial derivatives. The firs argument is the list of variables or parameters with respect to we want to perform the partial derivative. The next number is the maximum order of partial derivation. The next option may be `Until`, or `Only`, in case we want all partials until that order, or just the ones of order n. For example:
  `AddPartials->{{y,a},2, Until}` computes:

$$\frac{\partial x}{\partial y_0}, \frac{\partial x}{\partial a}, \frac{\partial y}{\partial y_0}, \frac{\partial y}{\partial a}, \frac{\partial z}{\partial y_0}, \frac{\partial z}{\partial a}, \frac{\partial^2 x}{\partial y_0^2}, \frac{\partial^2 x}{\partial a^2}, \frac{\partial^2 x}{\partial y_0 \partial a}, \frac{\partial^2 y}{\partial y_0^2}, \frac{\partial^2 y}{\partial a^2}, \frac{\partial^2 y}{\partial y_0 \partial a}, \frac{\partial^2 z}{\partial y_0^2}, \frac{\partial^2 z}{\partial a^2}, \frac{\partial^2 z}{\partial y_0 \partial a}.$$

Here we have some examples we can execute in our case:

```
CodeFiles[lorenz, "lorenzMinF",
      MinTIDES->"Fortran",
      Output->Screen]
```

to generate a minimal Fortran code with output on the screen. Or we can execute

```
CodeFiles[lorenz, "lorenzMP",
      Output->"lorenzMP.txt",
      PrecisionDigits->500,
      AddPartials->{{x, y, z, s}, 2, Until}]
```

if we want high precision integration (with 500) digits. In this case the output will be written in a file called `lorenzMP.txt`.

There are many other options and combinations which use is carefully described in the user guide within the software.

5 NUMERICAL EXPERIMENTS

In the numerical experiments we have used three kinds of architectures. Fortran integrations both in double and quadruple precision have been performed in a PC Pentium D 3.40 GHz. with Windows XP SP2. The high precision computation has been done in a PC Intel ® Core$^{\text{TM}}$ 2 Duo CPU E6750 2.66 GHz with Linux Kernel 2.6.27.7-smp compiled for i686. The compiler for Fortran has been `gcc 4.2` (`gfortran`) for double precision and `Lahey LF95` for quadruple precision. The compiler for C has been also `gcc 4.2`.

5.1 Test Problems

To illustrate the use of the integrator we present two classical problems in dynamical systems.

Arenstorf orbits [2]. These are the particular case of the planar orbits in the restricted three body problem. One considers two bodies of masses $1-\mu$ and μ, in circular rotation in a plane and a third body of negligible mass moving around in the same plane. The interval of integration is $t \in [0, 30]$. and the equations are:

$$
\begin{aligned}
&x'' = x + 2y' - \mu' \frac{x+\mu}{D_1} - \mu \frac{x-\mu'}{D_2},\\
&y'' = y - 2x' - \mu' \frac{y}{D_1} - \mu \frac{y}{D_2},\\
&\begin{cases}
D_1 = ((x+\mu)^2 + y^2)^{3/2}, \qquad D_2 = ((x-\mu')^2 + y^2)^{3/2},\\
x(0) = 0.994, \quad y(0) = 0,\\
x'(0) = 0, \quad y'(0) = -2.00158510637908252240537862224,\\
\mu = 0.012277471, \qquad \mu' = 1-\mu.
\end{cases}
\end{aligned}
$$

Lorenz Model [18]. The classical system given by equations (8) and the classical Salztman values of the paramenters: $r = 28$, $\sigma = 10$ and $b = 8/3$. We use the initial

conditions of an unstable periodic orbit up to 500 digits thanks to professor Divakar Viswanath [22].

$$\begin{aligned} x(0) &= -13.763610682134200\ldots \\ y(0) &= -19.578751942451795\ldots \\ z(0) &= 27, \\ T &= 1.5586522107161747\ldots \end{aligned}$$

5.2 Numerical tests

In the first test we are going to compare TIDES with `dop853` and `odex`. Since we have used the implementation of `dop853` and `odex` made by Hairer and Wanner [14] and it has been written in Fortran, we have compared them with the minf-tides engine, also in Fortran. We present in the figures relative error vs CPU time diagrams, a standard in numerical ODE community.

In the Figure 1 we have compiled all the programs using `gfortran` and the optimization option `-O2`, and allowing dense output. We can observe the same behavior for both for Arenstorf and Lorenz problems: For low precision demands `dop853` is the fastest integrator. However, when we achieve 10^{-10} precision level minf-tides becomes the fastest option. The reason is the variable order formulation of the Taylor series method which guarantees a softer growing slope. `odex` also has variable order formulation, but the extrapolation method needs more precision in order to beat the Runge-Kutta code.

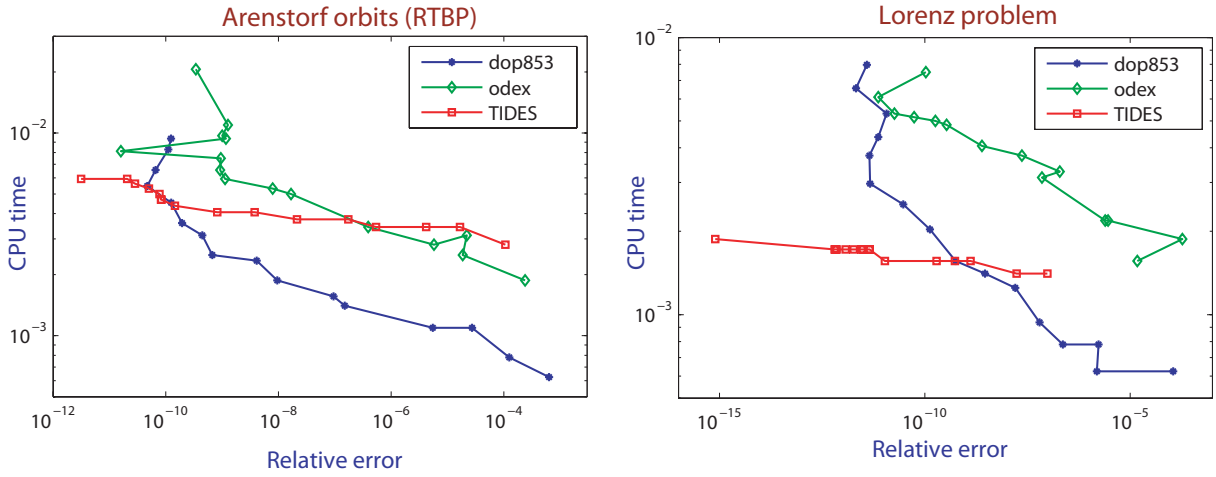

Figure 1.— Computational relative error vs CPU time diagrams in double precision

In Figure 2 we present the same tests, but we have used `Lahey LF95` Fortran compiler to have native quadruple precision. Since we can go farther than double precision we can appreciate that minf-tides and `odex` have almost the same slope, due to their variable

order formulation. We also can see that the difference between the variable order methods and Runge-Kutta is quite big for more accurate demands.

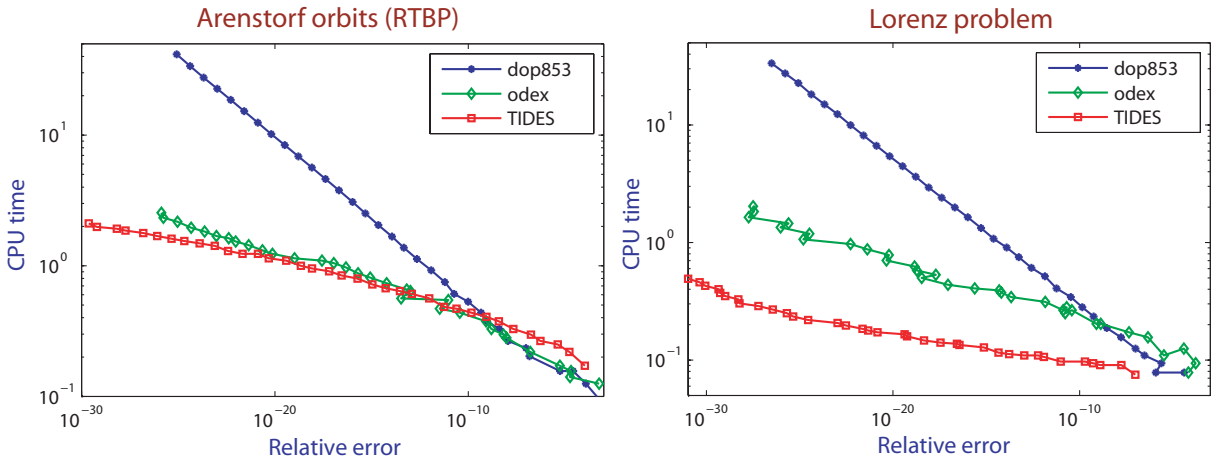

Figure 2.— Computational relative error vs CPU time diagrams in quadruple precision

High precision has become very useful both in theoretical and practical applications [3]. Our software includes a high precision engine, mp-tides, which can integrate differential equations (with partial derivatives if needed) with all the desired precision in a very easy way.

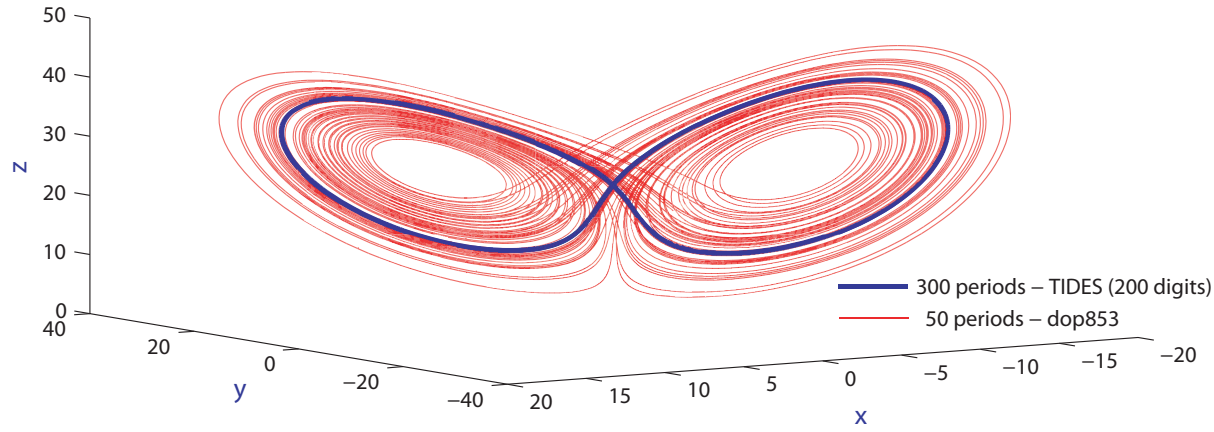

Figure 3.— 300 periods using mp-tides and 50 periods using `dopri853` in double precision

There are many problems in which, even if we are interested in results in double precision, the operations must be performed with enough digits of precision due to the unstable nature of the underlying system. The Lorenz system is a good example of system

of this nature. In Figure 3 we present integrations of the same orbit. We have drawn in red 300 revolutions using mp-tides with high precision arithmetic, and we can appreciate that the structure of the periodic orbit is preserved. In blue we have plotted only 50 periods performed with `dopri853` compiled and executed in double precision. In this case the loss of digits of precision is too fast, so with only a small number of periods, compared to the high precision computation, we have fallen into the famous chaotic attractor.

Acknowledgements

We thank Prof. Divarak Viswanath for providing us periodic initial conditions of the Lorenz system up to 500 digits. Also we thank the free software community for providing us arbitrary precision libraries of such an excellent quality. Finally we would like to remark that this work has been supported by the Spanish Research projects MTM2009-10767 and AYA2008-05572.

References

[1] R. Abad, R. Barrio, F. Blesa, and M. Rodríguez. TIDES: a Taylor series Integrator of Differential EquationS. *preprint*, 2010.

[2] R.F. Arenstorf. Periodic solutions of the restricted three body problems representing analytic continuations of keplerian elliptic motions. *Amer. J. Math.*, LXXXV:27–35, 1963.

[3] D.H. Bailey, J.M. Borwein, and R. Barrio. A High-Precision Computation: Mathematical Physics and Dynamics. *preprint*, 2009.

[4] R. Barrio, F. Blesa, and M. Lara. VSVO formulation of the Taylor method for the numerical solution of ODEs. *Comput. Math. Appl.*, 50(1-2):93–111, 2005.

[5] R. Barrio, F. Blesa, and S. Serrano. Bifurcations and safe regions in open Hamiltonians. *New J. Phys.*, 11:053004 (15pp.), 2009.

[6] R. Barrio and S. Serrano. A three-parametric study of the Lorenz model. *Phys. D*, 229(1):43–51, 2007.

[7] Roberto Barrio. Performance of the Taylor series method for ODEs/DAEs. *Appl. Math. Comput.*, 163(2):525–545, 2005.

[8] Roberto Barrio. Sensitivity analysis of ODEs/DAEs using the Taylor series method. *SIAM J. Sci. Comput.*, 27(6):1929–1947, 2006.

[9] M. Berz. COSY INFINITY version 8 reference manual. *Technical Report MSUCL—1088, National Superconducting Cyclotron Lab., Michigan State University, East Lansing, Mich.*, 1997.

[10] Martin Berz. Algorithms for higher derivatives in many variables with applications to beam physics. In *Automatic differentiation of algorithms (Breckenridge, CO, 1991)*, pages 147–156. SIAM, Philadelphia, PA, 1991.

[11] Y. F. Chang and G. Corliss. ATOMFT: solving ODEs and DAEs using Taylor series. *Comput. Math. Appl.*, 28(10-12):209–233, 1994.

[12] G. F. Corliss, A. Griewank, P. Henneberger, G. Kirlinger, F. A. Potra, and H. J. Stetter. High-order stiff ODE solvers via automatic differentiation and rational prediction. In *Numerical analysis and its applications (Rousse, 1996)*, volume 1196 of *Lecture Notes in Comput. Sci.*, pages 114–125. Springer, Berlin, 1997.

[13] Z. Galias and P. Zgliczyński. Computer assisted proof of chaos in the Lorenz equations. *Phys. D*, 115(3-4):165–188, 1998.

[14] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving ordinary differential equations. I*, volume 8 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, second edition, 1993.

[15] Àngel Jorba and Maorong Zou. A software package for the numerical integration of ODEs by means of high-order Taylor methods. *Experiment. Math.*, 14(1):99–117, 2005.

[16] Tomasz Kapela and Piotr Zgliczyński. The existence of simple choreographies for the N-body problem—a computer-assisted proof. *Nonlinearity*, 16(6):1899–1918, 2003.

[17] Tomasz Kapela and Piotr Zgliczyński. A Lohner-type algorithm for control systems and ordinary differential inclusions. *Discrete Contin. Dyn. Syst. Ser. B*, 11(2):365–385, 2009.

[18] E.N. Lorenz. Deterministic nonperiodic flow. *Journal of Atmospheric Sciences*, 20:130–141, 1963.

[19] N. S. Nedialkov, K. R. Jackson, and G. F. Corliss. Validated solutions of initial value problems for ordinary differential equations. *Appl. Math. Comput.*, 105(1):21–68, 1999.

[20] Nedialko S. Nedialkov and John D. Pryce. Solving differential algebraic equations by Taylor series. III. The DAETS code. *JNAIAM J. Numer. Anal. Ind. Appl. Math.*, 3(1-2):61–80, 2008.

[21] Louis B. Rall and George F. Corliss. An introduction to automatic differentiation. In *Computational differentiation (Santa Fe, NM, 1996)*, pages 1–18. SIAM, Philadelphia, PA, 1996.

[22] Divakar Viswanath. The fractal property of the Lorenz attractor. *Phys. D*, 190(1-2):115–128, 2004.

[23] Piotr Zgliczyński. C^1 Lohner algorithm. *Found. Comput. Math.*, 2(4):429–465, 2002.