TIDES user guide

(TIDES: a Taylor series Integrator of Differential EquationS)

A. Abad, R. Barrio, F. Blesa & M. Rodriguez

Grupo de Mecánica Espacial
Universidad de Zaragoza
50008 Zaragoza
Spain

e-mail: tides@unizar.es

`http://gme.unizar.es/software/tides`

Version 1.0:

Contents

Chapter 1

TIDES installation

1.1 What is TIDES?

The objective of TIDES (Taylor series Integrator of Differential EquationS) is the integration of systems of first order differential equations (ODEs),

$$\dot{\boldsymbol{y}} = \boldsymbol{f}(t, \boldsymbol{y}(t); \boldsymbol{p}), \quad \boldsymbol{y}(t_0) = \boldsymbol{y}_0, \quad \boldsymbol{y}\,(\text{variables}) \in \mathbb{R}^n, \quad \boldsymbol{p}\,(\text{parameters}) \in \mathbb{R}^m, \tag{1.1}$$

by using the Taylor series method. The Taylor series method (TSM) is based on the evaluation of the time Taylor series of the variables obtained by an iterative way that use the decomposition of the derivatives by automatic differentiation (AD)methods.

TIDES has two different parts (pieces of software): The Mathematica package MathTIDES and the C library LibTIDES.

TIDES		
Product		Language
MathTIDES	preprocessor	Mathematica (version 6.0 or greater)
LibTIDES	library (objects or source code)	C

The preprocessor MathTIDES writes, automatically, the files containing the code with the iterative scheme to obtain the Taylor Series of the variables. These files, together with the library LibTIDES, form the Taylor Series Method integrator (TSM Integrator). The multiple precision version of the integrator requires the MPFR library (`http://www.mpfr.org/`).

When you receive TIDES uncompress it in your home directory and then make the installation process.

1.2 How to install and use MathTIDES

To install MathTIDES you need to copy the folder MathTIDES inside a directory that is in the `$Path` of Mathematica. You can do this manually, or by opening the notebook `InstallMathTIDES.nb` that make automatic the installation process. Follow the installation instructions of `InstallMathTIDES.nb` and when MathTIDES has been installed load the package by writing

```
<<MathTIDES`
```

With `InstallMathTIDES.nb` you can also uninstall MathTIDES.

1.3 How to install LibTIDES

That follows install LibTIDES in a Unix system (Macos X and Windows with MinGW included). Let's suppose you have administrator privileges.

By default the LibTIDES library uses GMP and MPFR libraries for the multiple precision computations, so you need to have both installed. Then, you first need to install GMP and MPFR, in this order, if your system does not have them. You can download GMP from http://www.gmplib.org and MPFR from http://www.mpfr.org. Then you must uncompress them and run on the terminal the following four orders

```
./configure
make
make check
sudo make install
```

inside each of its directories.

The complete installation of LibTIDES follows the same steps: uncompress the files and run on the terminal the previous four orders inside the directory of TIDES. The installation process changes if you do not want the complete set of options of TIDES or you have not administrator privileges.

1.3.1 Configuring the installation

To configure the complete installation type on the terminal

```
./configure
```

Depending on where you installed GMP and/or MPFR, you may need to specify its installation directories. For example, if you put GMP in /usr/local, then you need to do the following

```
./configure --with-gmp=/usr/local
```

If MPFR is also in a non-standard directory, you may have to do the same thing with it:

```
./configure --with-gmp=/usr/local --with-mpfr=/usr/local
```

If you don't have GMP and/or MPFR installed, or you are not interested in having multiple precision capacities in your program, you have to pass the following option to configure:

```
./configure --disable-multiple-precision
```

This will create the needed Makefiles to compile a reduced version of `libTIDES.a` without the MPFR extensions.

By default, the library is installed in `/usr/local/lib`. If you prefer another installation directory, you have to specify it by adding the prefix option to configure.

```
./configure --disable-multiple-precision --prefix=......
```

After this installation process, you will end with a library containing the objects needed for the standard and multiple precision TIDES packages.

1.3.2 Making the library

To build the library, type on the terminal:

```
make
```

This will create the complete library or only with the double precision version of the library depending on the options of `configure`

1.3.3 Checking the library

Before to install LibTIDES it is useful to check the created library. To check the build library (run the test files), type:

```
make check
```

The test includes the double precision test and the multiple precision test when available. If you have a FORTRAN compiler and you plan to use the minimal FORTRAN version of TIDES you can pass FORTRAN tests by adding one option to the configuration

```
./configure --enable-fortran-tests
```

If everything is OK, you can install it.

1.3.4 Installing and uninstalling the library

If you have administrator privileges you can install the library by typing on the terminal

```
sudo make install
```

To uninstall the library just type

```
sudo make uninstall
```

The word `sudo` it is not necessary if you are `root user`.

If you have not administrator privileges take the library `libTIDES.a` created on the TIDES directory and copy it on your desired directory.

1.3.5 Working with Mac OS X

Taking into account that Mac OS X is based on a Unix system you can install LibTIDES on Mac OS X by following all the previous steps from the terminal, and using the `gcc` compiler installed on Mac OS X with the Developer tools.

If you prefer to use XCode, instead working from the terminal, follow the previous steps except the order `sudo make install`. Then take the library `libTIDES.a` and include it in your XCode project. If you work with multiple precision do the same with MPFR and GMP libraries.

1.3.6 Working with Windows

The installation has been tested with `MinGW` and `Msys`. The GMP and MPFR libraries are not installed, so you have to build and install them. They will be installed at `/usr/local`, but `Msys` does not have it in the path, so you have to use:

```
./configure --with-gmp=/usr/local --with-mpfr=/usr/local
```

to correctly create the Makefiles needed for building LibTIDES.

Chapter 2

Options in TIDES

In this section we present the different options in the construction of the TSM Integrator, and the options of the TSM Integrator itself. These options are highlighted with a gray or yellow `color box`.

2.1 Four versions of the TSM Integrator

With the preprocessor MathTIDES we may write four different versions of the TSM Integrator. Two minimal (faster) versions in FORTRAN (`minf-tides`) and C (`minc-tides`) respectively, and two standard (more complete) versions in C, with double (`dp-tides`) or arbitrary precision (`mp-tides`) respectively.

Version	Contents	MathTIDES generates	linked with
minf-tides	basic TSM	FORTRAN files	
minc-tides	basic TSM	C files	
dp-tides	complete TSM + partial derivatives	C files	LibTIDES
mp-tides	complete TSM + partial derivatives + arbitrary precision	C files	LibTIDES MPFR library

2.1.1 Minimal versions (minf-tides, minc-tides)

The minimal versions of the TSM Integrator produce a basic Taylor series integrator characterized by the following points

- In the mathematical expression of $\boldsymbol{f}$ in (1.1) may appear the following functions:
  - The usual operators: $+, -, *, /$
  - A number(or constant parameter) power to a variables: $a^x, a > 0$.
  - A variable power to a number(or constant parameter): $x^r, r \in \mathbb{R}$.
  - Functions: $\sin, \cos, \tan, \log$.
- They integrate only one differential system on each main problem.
- They write the output, dense or not, into a file or on the screen.

minf-tides is based on three FORTRAN files generated by MathTIDES. One file contains the iterative procedure to construct the function $\boldsymbol{f}$. The second file, whose name begins by `dr_`, contains the driver (main program) to call to the core of the integrator. The third file, named `minf_tides.f`, is always the same and contains the kernel of the integrator.

minc-tides is based on four C files generated by MathTIDES. Two files with the same names and extensions `.c, .h` contains the iterative procedure to construct the function $\boldsymbol{f}$. The third file, whose name begins by `dr_`, contains the driver (main program) to call to the core of the integrator. The fourth file, named `minc_tides.c`, is always the same and contains the kernel of the integrator.

To integrate the ODE we only need to compile and run these files and it is not necesary to link the files together with the library LibTIDES.

2.1.2 Standard versions (**dp-tides, mp-tides**)

The standard versions of the integrator produce a complete Taylor series integrator characterized by the following points

- In the mathematical expression of $\boldsymbol{f}$ in (1.1) may appear the following functions:
  - The usual operators: $+, -, *, /$
  - A number(or constant parameter) power to a variables: $a^x, a > 0$.
  - A variable power to a number(or constant parameter): $x^r, r \in \mathbb{R}$.
  - Functions: $\sin, \cos, \tan, \sinh, \cosh, \tanh, \mathrm{asin}, \mathrm{acos}, \mathrm{atan}, \mathrm{asinh}, \mathrm{acosh}, \mathrm{atanh}, \log$.
- They integrate one or more differential systems on each main problem.
- They write the output, dense or not, into a file or on the screen and/or a bidimensional array.
- Simultaneously with the integral of the variables they may obtain :
  - The integral of functions of the variables.
  - The integral of the partials of the variables with respect to the initial conditions.
  - The integral of the partials of the variables with respect to the parameters.
  - The integral of the partials of functions of the variables with respect to the initial conditions.
  - The integral of the partials of functions of the variables with respect to the parameters.

Both standard versions are based on three C files: the driver (basic main program) an two files, with the same names and extensions `.c, .h`, that contains the iterative procedure to construct the function $\boldsymbol{f}$. These files must be compiled and linked with the library LibTIDES (kernel of the integrator) to integrate the ODE.

mp-tides uses the MPFR library (**libmpfr.a**) to integrate in multiple precision with any number of precision digits.

2.2 Parameters of the Taylor Series Method (TSM)

Let us consider the initial value problem:

$$\frac{d\boldsymbol{y}(t)}{dt} = \boldsymbol{f}(t, \boldsymbol{y}(t); \boldsymbol{p}), \qquad \boldsymbol{y}(t_0) = \boldsymbol{y}_0, \qquad t \in \mathbb{R},\, \boldsymbol{y} \in \mathbb{R}^n,\, \boldsymbol{p} \in \mathbb{R}^m \tag{2.1}$$

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 consider that $\boldsymbol{f}$ is analytic).

$$\begin{aligned}
\boldsymbol{y}(t_0) &\stackrel{\text{def}}{=} \boldsymbol{y}_0, \\
\boldsymbol{y}(t_{i+1}) &\simeq \boldsymbol{y}(t_i) + \frac{d\boldsymbol{y}(t_i)}{dt}\, h_{i+1} + \frac{1}{2!}\frac{d^2\boldsymbol{y}(t_i)}{dt^2}\, h_{i+1}^2 + \ldots + \frac{1}{N!}\frac{d^N\boldsymbol{y}(t_i)}{dt^N}\, h_{i+1}^N \\
&\simeq \boldsymbol{y}_i + \boldsymbol{f}(t_i,\boldsymbol{y}_i)\, h_{i+1} + \frac{1}{2!}\frac{d\boldsymbol{f}(t_i,\boldsymbol{y}_i)}{dt}\, h_{i+1}^2 + \ldots + \frac{1}{N!}\frac{d^{N-1}\boldsymbol{f}(t_i,\boldsymbol{y}_i)}{dt^{N-1}}\, h_{i+1}^N \stackrel{\text{def}}{=} \boldsymbol{y}_{i+1}.
\end{aligned} \tag{2.2}$$

From the formulation of the TSM, the problem is reduced to the determination of the Taylor coefficients $\{d^j\boldsymbol{y}(t_i)/dt^j\}$ by means of the use of automatic differentiation (AD) techniques.

The TSM presents several peculiarities. One of them is that it gives directly a dense output in the form of a power series and therefore we can evaluate the solution at any time just by using the Horner algorithm. Also, as TSM of degree N are also of order N, the use of TSMs of high degree give us numerical methods of high order. Therefore, they are very useful for high-precision solution of ODEs.

In the practical implementation of a numerical method for the solution of ODEs the use of variable stepsizes is a crucial point because it permits to automatize the control of the error. In TIDES we use an absolute error tolerance `tolabs` and a relative tolerance `tolrel`. With both we construct the error tolerance

$$\texttt{TOL} = \boxed{\texttt{tolabs}} + \max(\|\boldsymbol{y}(t_i)\|, \|\boldsymbol{y}(t_{i-1})\|) \times \boxed{\texttt{tolrel}}$$

Another crucial point in the TSM is the selection of the order of the method, that is, N. In TIDES we adopt a modification of the *optimal order*. On one hand, when we use an order that depends only on the requested tolerance `tolabs`, we adopt the simple formula

$$\hat{n} = \lceil -\ln(\texttt{tolabs})/2 \rceil + \boxed{\texttt{nordinc}}$$

where `maxord` is the maximum order and `nordinc` is an increment of the order with respect to the asymptotic formula (this may be adjusted by the user). This is the case on the dp-tides and mp-tides programs, where the complexity of the extended Taylor series algorithm does not justify to use a more adaptive algorithm. In the minf-tides and minc-tides programs we use a slightly more sophisticated formula

$$\begin{aligned}
\texttt{tolorder}(i) &= \min\left(\texttt{tolabs}/\min(\|\boldsymbol{y}(t_i)\|, \|\boldsymbol{y}(t_{i-1})\|), \texttt{tolrel}\right), \\
\hat{n} &= \lceil -\ln(\texttt{tolorder}(i))/2 \rceil + \texttt{nordinc}.
\end{aligned}$$

In both cases we use

$$N = \max\left(\boxed{\texttt{minord}}, \hat{n}\right).$$

We use two strategies for selecting the stepsize. The first one is based on estimating the error just by taking the last term in the Taylor series (in order to avoid problems with odd/even functions we take the last two terms different from zero, which avoid also problems with polynomial solutions). Note that this strategy is also equivalent to the concept of RK pairs (two RK methods, one of lower order than the other, which permits to estimate the error). So,

$$\begin{aligned}
\hat{h}_{i+1} &= \min\left\{ \left(\frac{\texttt{TOL}}{\|\boldsymbol{y}^{[N-1]}(t_i)\|_\infty}\right)^{1/(N-1)}, \left(\frac{\texttt{TOL}}{\|\boldsymbol{y}^{[N]}(t_i)\|_\infty}\right)^{1/N} \right\}, \\
h_{i+1} &= \boxed{\texttt{fac1}} \times \max\left(\min(\boxed{\texttt{rmaxstep}} \times h_i, \hat{h}_{i+1}), \boxed{\texttt{rminstep}} \times h_i\right),
\end{aligned} \tag{2.3}$$

with $\boldsymbol{y}^{[N]}$ the normalised derivative $\boldsymbol{y}^{[N]} = \boldsymbol{y}^{(N)}/N!$, `fac1` a safety factor (we use $\texttt{fac1} = 0.9$), and `rmaxstep` and `rminstep` stands for the maximum and minimum ratio between the actual stepsize and the previous one.

After this selection of the stepsize we may enter, or not, in a refinement process which is based on the `defect error control`. Note that, "a priori", in the TSM there is no rejected step as occurs in any variable-stepsize formulation for Runge-Kutta or multistep methods because we choose the stepsize once the series are generated in order to obtain a required precision level. But, in order to give more guarantee about the stepsize we may analyse the agreement between the tangent vector to the Taylor polynomial and the vector field at the end of the step, that is, given the Taylor approximation of the solution on the interval $[t_i,\, t_{i+1}] = [t_i,\, t_i + h_{i+1}]$

$$\boldsymbol{y}(t) \simeq \sum_{k=0}^{N} \boldsymbol{y}^{[k]}(t_i) \cdot (t - t_i)^k, \qquad \boldsymbol{y}'(t) \simeq \sum_{k=1}^{N} k\, \boldsymbol{y}^{[k]}(t_i) \cdot (t - t_i)^{k-1}$$

then evaluating at the end of the interval $\boldsymbol{y}'_{i+1} \equiv \sum_{k=1}^{N} k\, \boldsymbol{y}^{[k]}(t_i) \cdot (h_{i+1})^{k-1}$ and the criteria for rejecting the stepsize is

$$\text{if} \quad \|\boldsymbol{y}'_{i+1} - \boldsymbol{f}(t_{i+1},\, \boldsymbol{y}_{i+1})\|_\infty > \texttt{fac2} \times \texttt{TOL} \quad \text{then} \quad \widetilde{h}_{i+1} = \texttt{fac3} \cdot h_{i+1}, \tag{2.4}$$

where `fac2` and `fac3` are control factors that reduces the stepsize (we have taken $\texttt{fac2} = 10$, $\texttt{fac3} = 0.8$). It is important to remark that although the stepsize may be rejected we do not have to recalculate the Taylor coefficients, we only have to consider the new stepsize and enter again in the criteria for rejecting the stepsize. Therefore we cannot say that we reject a complete step, we just reject the estimation of the stepsize, and so the computational cost is not very hight (in fact the cost of evaluating $\boldsymbol{y}'_{i+1}$ and $\boldsymbol{f}(t_{i+1},\, \boldsymbol{y}_{i+1})$). This process is done a maximum of `nitermax` times.

2.3 Inputs and Outputs of the TSM Integrator

To integrate the ODE (1.1) the TSM Integrator needs the numerical value of the initial conditions of the variables $\boldsymbol{y}_0$ and the numerical value of the parameters $\boldsymbol{p}$. These values must be passed to the TSM Integrator as the main input.

We may choose between a dense output, i.e. the solution in a list of equidistant (or not) points $\{t_0, t_1, \ldots, t_f\}$, or a non dense output, i.e. only at the final point t_f.

The basic output of a TSM Integrator is the result of the integration, i.e. the value of the variables $\boldsymbol{y}(t)$ in the desired points: $\{t_0, t_1, \ldots, t_f\}$. Likewise, we may add to the output the values of a function $G(\boldsymbol{y}(t))$ and the values of the partials of $\boldsymbol{y}(t)$ and $G(\boldsymbol{y}(t))$ with respect to the initial conditions or the parameters evaluated in the same points $\{t_0, t_1, \ldots, t_f\}$.

The previous output has the format of a matrix in which each row i represent the solution in t_i. The elements of the row are: $t_i, \boldsymbol{y}(t_i)$, and depending on the case, $G(\boldsymbol{y}(t_i))$, $\partial \boldsymbol{y}(t_i)/\partial s_j$, $\partial G(\boldsymbol{y}(t_i))/\partial s_j$, with s_i the elements, in order with respect to we compute the partials.

The output can be written on a data matrix (only in the standard versions) and into a file or the screen (all versions).

We may summarise the options of a TSM Integrator, to obtain the desired solution, in the following scheme

- A vector with the `initial conditions` $\boldsymbol{y}_0$.
- A vector with the `parameters` $\boldsymbol{p}$.
- The list $\{t_0, t_1, \ldots, t_f\}$ of `integration points` for the dense output or the initial and the final point $\{t_0, t_f\}$ for a non dense output.
- The way in which we want the output: `file`, `screen` or `data matrix`.

Chapter 3

How to use MathTIDES

3.1 Representing ODEs in MathTIDES

The Taylor Series Method integrates only ODE systems of first order. However, a higher order ODE, with certain conditions, may be transformed into a first order ODE. Applying the Newton's equations to a potential function or the Hamilton's equations to a Hamiltonian we also obtain first order ODEs.

In MathTIDES a first order ODE is represented by an expression with head `FirstOrderODE$`. However, the user will declare the ODE with an expression with one of the following heads:

- `FirstOrderODE` : declares a first order ODE directly.
- `NthOrderODE` : declares a first order ODE from a k-th order ODE.
- `PotentialToODE` : declares a first order ODE from a potential function V.
- `HamiltonianToODE` : declares a first order ODE from a hamiltonian function $\mathcal{H}$.

The result in all cases is an expression with head `FirstOrderODE$` that contains the internal representation in MathTIDES of a first order differential equation.

3.1.1 First order differential equations

A first order ODE is represented by the equation

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}(t, \boldsymbol{y}(t); \boldsymbol{p}), \quad \boldsymbol{y}(t_0) = \boldsymbol{y}_0, \quad \boldsymbol{y} \in \mathbb{R}^n, \quad \boldsymbol{p} \in \mathbb{R}^m, \tag{3.1}$$

where

- t is the independent variable. It may appear explicitely or not.
- $\boldsymbol{y} = (y_1, \ldots, y_n)$ is the n-dimensional vector of variables ($n > 0$).
- $\boldsymbol{p} = (p_1, \ldots, p_m)$ is the m-dimensional vector of parameters ($m \geq 0$).
- $\boldsymbol{f} = (f_1, \ldots, f_n)$ is the n-dimensional vector of functions (expressions) representing the first order derivatives of the variables.

To declare a first order differential equation we will use an expression with the head `FirstOrderODE` and the following arguments and options:

- *First argument:* the list of the expressions $\{f_1, \ldots, f_n\}$ of the derivatives of the variables. The number n of elements of the list must be equal to the number of variables. If n = 1 the argument is not a list.
- *Second argument:* the symbol that represents the independent variable t. This symbol may appear explicitely or not in the first argument.
- *Third argument:* the list $\{y_1, \ldots, y_n\}$ of symbols that represent the variables. It has the same number of elements than the first argument. If $n = 1$ the argument is not a list.
- *Fourth argument:* the list $\{p_1, \ldots, p_m\}$ of symbols that represent the parameters. If the number of parameters m is equal to 1 the argument is not a list. If there is no parameter ($m = 0$) this argument may be avoided.

To illustrate the use of `FirstOrderODE` let us take two examples. The first one is the system of equations

$$\frac{dx}{dt} = y, \quad \frac{dy}{dt} = -x, \tag{3.2}$$

whose solution for $x(0) = 0$, $y(0) = 1$ gives the functions: $x(t) = \sin t$, $y(t) = \cos t$. To declare this ODE we will write the expression

In[1]:=

```
sincos = FirstOrderODE[{y, -x}, t, {x, y}]
```

Out[1]=

```
FirstOrderODE$[{y, -x}, t, {x, y}, {}, {}]
```

The second example is the equation that define, for the initial condition $x(0) = 0$, the elliptic integral of the first kind

$$\frac{dx}{dt} = \frac{1}{\sqrt{1 - k^2 \sin^2 t}}, \tag{3.3}$$

that we declare with the expression

In[2]:=

```
ellF = FirstOrderODE[1/Sqrt[1 - k^2 Sin[t]^2], t, x, k]
```

Out[2]=

```
FirstOrderODE$[{1/Sqrt[1 - k^2 Sin[t]^2]}, t, {x}, {k}, {}]
```

where the modulus k acts as a parameter.

3.1.2 Higher order differential equations

Let us consider an ODE system represented by means of the expressions

$$\boldsymbol{F}(t, \boldsymbol{y}, \frac{d\boldsymbol{y}}{dt}, \frac{d^2\boldsymbol{y}}{dt^2}, \ldots, \frac{d^k\boldsymbol{y}}{dt^k}; \boldsymbol{p}) = 0, \qquad \boldsymbol{y}(t_0) = \boldsymbol{y}_0, \ldots, \frac{d^k\boldsymbol{y}}{dt^k}(0) = \boldsymbol{y}_0^{(k)}, \tag{3.4}$$

where $\boldsymbol{F}, \boldsymbol{y} \in \mathbb{R}^n$, and $\boldsymbol{p} \in \mathbb{R}^m$.

If all the derivatives $y_1^{(k)}, \ldots y_n^{(k)}$ of the greatest order k appears explicitely in (3.4), then, solving the system (3.4) in $y_1^{(k)}, \ldots y_n^{(k)}$, if it is possible, we can transform the k-th order ODE into a first order ODE by introducing the derivatives $\frac{d\boldsymbol{y}}{dt}, \frac{d^2\boldsymbol{y}}{dt^2}, \ldots, \frac{d^{k-1}\boldsymbol{y}}{dt^{k-1}}$ as new variables of the system.

MathTIDES tranforms automatically a k-th order ODE into a first order ODE by using an expression with head `NthOrderODE` and the following arguments

- *First argument:* the list of the expressions $\{F_1, \ldots, F_n\}$ that represent the system of equations with a format defined by the following rules:
  - The derivatives of a variable of symbol `x` must be represented by quotes: `x, x', x'', x'''`, ..
  - The equations are represented by means of the symbol ==
  - The number of equations is equal to the number of variables.
  - If the number of variables is equal to one, the first and the third arguments are not lists.
  - In the system it must appear the derivatives of greater order of all the variables.
- *Second argument:* the symbol that represents the independent variable t. This symbol may appear explicitely or not in the first argument.
- *Third argument:* the list $\{y_1, \ldots, y_n\}$ of symbols that represent the variables. It has the same number of elements than the first argument. If $n = 1$ the argument is not a list.
- *Fourth argument:* the list $\{p_1, \ldots, p_m\}$ of symbols that represent the parameters. If the number of parameters m is equal to 1 the argument is not a list. If there is no parameter ($m = 0$) this argument may be avoided.

A k-th order differential equation is transformed into an equivalent system of first order differential equations by extending the number of variables. If a variable have the symbol `xxx`, the derivatives of this variable are converted in new variables whose symbol has the same beginning `xxx` and ends by `$di`, with `i` the order of the variable:

```
x'   ---> x$d1
x''  ---> x$d2
x''' ---> x$d3
```

The order of the variables of the final system of equation is the following:

1. Variables (in the same order that before)
2. First derivatives (mantaining the relative order of the variables)
3. Second derivatives (mantaining the relative order of the variables)
4. ....

To illustrate the use of `NthOrderODE` let us take two examples. The first one is the harmonic oscillator

$$\frac{d^2x}{dt^2} + \omega x = 0. \tag{3.5}$$

To declare this differential equation in MathTIDES we will write the expression

---

In[3]:=

```
oscillator = NthOrderODE[x'' + w x == 0, t, x, w]
```

Out[3]=

```
FirstOrderODE$[{x$d1, -x w}, t, {x, x$d1}, w}, {}]
```

Let us observe the list of variables {`x, x$d1`} of the transformed system.

The second example we present the following third order ODE

$$\begin{aligned} x''' - 2y'' + x' &= 2x^2 - y, \\ 4y''' - 2x''y' &= 2x + y^2. \end{aligned}$$

In MathTIDES we will write

In[4]:=

```
ntheq = NthOrderODE[
          {x''' - 2 y'' + x ' == 2 x^2 - y,
          4 y''' - 2 x '' y' == 2 x + y^2}, t, { x, y}]
```

Out[4]=

```
FirstOrderODE$[{x$d1, y$d1, x$d2, y$d2, 2 x^2 - x$d1 - y + 2 y$d2,
  1/4 (2 x + y^2 + 2 x$d2 y$d1)}, t, {x, y, x$d1, y$d1, x$d2,
  y$d2}, {}, {}]
```

Let' s observe again the list of variables {`x, y, x$d1, y$d1, x$d2, y$d2`} of the transformed system.

3.1.3 From potential to Newton's equations

Let' s suppose a potential $V(\boldsymbol{y}, \boldsymbol{p})$ in the variables $\boldsymbol{y} \in \mathbb{R}^n$, and with m parameters $\boldsymbol{p} \in \mathbb{R}^m$, then the Newton's equations $\ddot{\boldsymbol{y}} = -\nabla V(\boldsymbol{y}, \boldsymbol{p})$ will be obtained as a first order ODE by means of the MathTIDES expression of head `PotentialToODE` that have the following arguments:

- *First argument:* the expression of the potential V. This expression is never a list.
- *Second argument:* the symbol that represents the independent variable t. This symbol does not appear in the potential function.
- *Third argument:* the list $\{y_1, \ldots, y_n\}$ of symbols that represent the variables. If $n = 1$ the argument is not a list.

- *Fourth argument:* the list $\{p_1, \ldots, p_m\}$ of symbols that represent the parameters. If the number of parameters m is equal to 1 the argument is not a list. If there is no parameter ($m = 0$) this argument may be avoided.

As an example let us take the Keplerian problem, in which the potential is given by

$$V = \frac{\mu}{\sqrt{x^2 + y^2 + z^2}}, \tag{3.6}$$

where μ represents a parameter.

In[5]:=

```
PotentialToODE[-mu/Sqrt[x^2 + y^2 + z^2], t, {x, y, z}, mu]
```

Out[5]=

```
FirstOrderODE$[{x$d1, y$d1,
  z$d1, -((mu x)/(x^2 + y^2 + z^2)^(3/2)), -((
   mu y)/(x^2 + y^2 + z^2)^(3/2)), -((mu z)/(x^2 + y^2 + z^2)^(
   3/2))}, t, {x, y, z, x$d1, y$d1, z$d1}, {mu}, {}]
```

`PotentialToODE` computes the gradient of the potential and transforms the second order Newton's equation into a first order equation duplicating the number of variables `{x, y, z, x$d1, y$d1, z$d1}`.

3.1.4 Hamilton's equations

Let' s suppose a dynamical system described by a Hamiltonian $\mathcal{H}(t, \boldsymbol{x}, \boldsymbol{X}, \boldsymbol{p})$ where t is the independent variable (it may appear explicitely or not), $\boldsymbol{x}$ is the n-dimensional vector of variables, $\boldsymbol{X}$ is the n-dimensional vector of associated momenta and $\boldsymbol{p}$ is the m-dimensional vector of parameters. Then the first order ODE that represents the dynamical system is given by the Hamilton's equations

$$\frac{d\boldsymbol{x}}{dt} = \frac{\partial \mathcal{H}}{\partial \boldsymbol{X}}, \quad \frac{d\boldsymbol{X}}{dt} = -\frac{\partial \mathcal{H}}{\partial \boldsymbol{x}}. \tag{3.7}$$

With MathTIDES we create the differential equations directly from the Hamiltonian by using an expression with the head `HamiltonianToODE` and the following arguments and options:

- *First argument:* the expression of the Hamiltonian $\mathcal{H}$. This expression is never a list.
- *Second argument:* the symbol that represents the independent variable t. This symbol may appear or not in the hamiltonian.
- *Third argument:* the list $\{x_1, \ldots, x_n, X_1, \ldots X_n\}$ of symbols that represent the variables and momenta. The length of this list is always an even number. The order of the momenta corresponds with the order of the associated variables.
- *Fourth argument:* the list $\{p_1, \ldots, p_m\}$ of symbols that represent the parameters. If the number of parameters m is equal to 1 the argument is not a list. If there is no parameter ($m = 0$) this argument may be avoided.

As an example we take the planar keplerian problem whose hamiltonian is given by the expression

$$\mathcal{H} = \frac{X^2 + Y^2}{2} - \frac{\mu}{\sqrt{x^2 + y^2}}, \tag{3.8}$$

where the variables (x, y) represent the position, the momenta (X, Y) represent the velocity and μ represents a parameter.

In[6]:=

```
hamkep = HamiltonianToODE[(X^2 + Y^2) /2 -mu/Sqrt[x^2 + y^2],
   t, {x, y, X, Y}, mu]
```

Out[6]=

```
FirstOrderODE$[{X,
   Y, -((x mu)/(x^2 + y^2)^(3/2)), -((y mu)/(x^2 + y^2)^(
   3/2))}, t, {x, y, X, Y}, {mu}, {}]
```

3.2 Creating the TSM Integrator with MathTIDES

3.2.1 How to create the TSM Integrator

To create the C or FORTRAN code to use together with the TIDES library we will use an expression with head `CodeFiles` and the following arguments:

- *First argument:* the first order differential equation. This is an expression with head `FirstOrderODE$` created by one of the previously described expressions.
- *Second argument:* an string that represents name of the files. With this name MathTIDES writes several files (depending on the options) with extension `.h`, `.c` or `.f`
- *Options:* the options and their default values(the value the MATHEMATICA takes when the option does not appear) are :

In[7]:=

```
Options[CodeFiles]
```

Out[7]=

```
{PrecisionDigits -> 16, MinTIDES -> False, Driver -> True,
 OnlyDriver -> False, ParametersValue -> Null,
 InitialConditions -> Null, IntegrationPoints -> Null,
 Output -> False, DataMatrix -> False, Factor1 -> Null,
 Factor2 -> Null, Factor3 -> Null, MaxStepRatio -> Null,
 MinStepRatio -> Null, MaxIterationsNumber -> Null,
 OrderIncrement -> Null, MinOrder -> Null,
 RelativeTolerance -> Null, AbsoluteTolerance -> Null,
 DefectErrorControl -> False}
```

3.2.2 Files created with `CodeFiles`

Let' s suppose that we write `"name"` as the second argument of `CodeFiles`. Then `CodeFiles` writes the following files:

- Minimal Version in C (minc-tides)
  - A driver (main program) named `"dr_name.c"`.
  - A file `"name.c"` with the differential equation.
  - Two files `"minc_tides.h"` and `"minc_tides.c"` with the kernel of the TSM Integrator.
  - Compiling and running the three files with extension `.c` we integrates the differential equation.
- Minimal Version in FORTRAN (minf-tides)
  - A driver (main program) named `"dr_name.f"`.
  - A file `"name.f"` with the differential equation.
  - A file `"minf_tides.f"` with the kernel of the TSM Integrator.
  - Compiling and running the three files with extension `.f` we integrates the differential equation.
- Standard versions (dp-tides and mp-tides)
  - A driver named `"dr_name.c"`.
  - Two files `"name.h"` and `"name.c"` with the differential equation.
  - Compiling `"dr_name.c"` and `"name.c"` and linking them with LibTIDES (and **libmpfr.a** with the version mp-tides) we obtain the executable to integrate the ODE.

The files written by MathTIDES are saved on the default directory of Mathematica (that obtained with `Directory[]`).

The user may change the default directory by using `SetDirectory`. For instance to change the default directory to the directory where the local Mathematica notebook is, use the expression

In[8]:=

```
SetDirectory[NotebookDirectory[]];
```

The expression `CodeFiles` shows on the screen the names of the created files and the directories where they had been stored.

3.2.3 Options to change the version and the files written by TIDES

Option `MinTIDES`

`MinTIDES` is used to create files to use with the minimum versions of TIDES. Use `MinTIDES -> "C"` to create the C minimum version minc-tides and `MinTIDES -> "Fortran"` to create the FORTRAN minimum version minf-tides.

Option `PrecisionDigits`

By default, when the option `MinTIDES` is not used an standard version is created. We choose between dp-tides or mp-tides by means of the option `PrecisionDigits`.

By default this option has the value `PrecisionDigits->16`. This means that the standard double precision version dp-tides is created. With a number greater than 16 this option declares the number of digits of precision of the TSM Integrator and creates the multiple precision version mp-tides.

Option `Driver`

By default a driver with the main program is created. With the option `Driver -> False`, `CodeFiles` does not write a driver, but it writes the rest of the files.

Option `OnlyDriver`

The option `OnlyDriver -> True` creates only the driver with the main program, and no other file.

3.2.4 Options to change how to call to the integrator in the driver

Option `InitialConditions`

With the option `InitialConditions -> {0.1, -2.3, ...}` we change, on the driver, the initial value of the vector of variables. The length of the list must be equal to the number of variables. If we do not use this options stars, `******`, instead of values appear on the driver.

Option `ParametersValue`

With the option `ParametersValue -> {0.1, -2.3, ...}` we change, on the driver, the value of the parameters. The length of the list must be equal to the number of parameters. If we do not use this options stars, `******`, instead of values appear on the driver.

Option `IntegrationPoints`

With this option we declare, on the driver, the list of points in which the solution is computed There are several versions of this option:

- `IntegrationPoints -> {t0, t1, ..., tf}`
  - `t0` is the initial integration point (where the initial conditions are given). It is a real number.
  - `t1,...,tf` are the points where we want to compute the solution. They all are real numbers. `tf` is the final integration point.
  - This option is only valid for the standard versions. In minimal versions you can use `IntegrationPoints -> {t0, tf}`, with the initial and final point, for non-dense output.
  - `{t0, t1, ..., tf}` are in order (crescent or decrescent). They can be non-equidistant points.
- `IntegrationPoints -> {t0, tf, Delta[dt]}`
  - `t0` is the initial integration point (real number).
  - `tf` is the final integration point (real number). It can be lesser or greater than `t0`.
  - `dt` is the interval between points in dense output (real number). If `tf` is lesser than `t0`, it must be negative.
  - The solution is computed in $\{t_0, t_1, \ldots, t_k\}$ = `{t0, t0+dt, t0+2*dt, ...  t0+k*dt}`, with `k` such us `t0+k*dt <= tf < t0+(k+1)*dt`. Not always the last point of the dense output coincides with the end integration point `tf`.
- `IntegrationPoints -> {t0, tf, Points[k]}`
  - `t0` is the initial integration point (real number).
  - `tf` is the final integration point (real number). It can be lesser or greater than `t0`.
  - `k` is an integer with the number of equidistant points in which the solution is computed. `dt` for dense output is equal to `(tf-t0)/k`.

    - The solution is computed in $\{t_0, t_1, \ldots, t_k\}$ = {`t0, t0+dt, t0+2*dt, ..., t0+k*dt = tf`}.

- `IntegrationPoints -> {t0, Delta[dt], Points[k]}`
    - `t0` is the initial integration point (real number).
    - `dt` is the interval between points in dense output (real number). It can be positive or negative.
    - `k` is an integer with the number of equidistant points in which the solution is computed.
    - The solution is computed in $\{t_0, t_1, \ldots, t_k\}$ = {`t0, t0+dt, t0+2*dt, ..., t0+k*dt`}.

Options `RelativeTolerance` and `AbsoluteTolerance`

Declares the value of the tolerances in the application of the method.

```
RelativeTolerance -> rtol
AbsoluteTolerance-> atol
```

`rtol` and `atol` are real numbers. The default value is 10^{-p}, where p is the value of the option `PrecisionDigits` for both tolerances. If only one tolerance is declared both are taken equals.

3.2.5 Options to change the ODE

Option `Optimization` (optimizing the linked functions)

With the default option `Optimization-->1`, MathTIDES uses the function `Simplify` to simplify the linked function used to apply the Taylor method to the ODE. With `Optimization-->2` MathTIDES tries to simplify with `FullSimplify`, and with `Optimization-->0` no simplification is made. The option `Optimization-->2` not ensure a drastic simplification, with respect the default but, sometimes, it takes a very long time of computation.

Option `AddPartials` (adding partial derivatives to the differential equations)

Together with the time evolution of the variables and functions we may compute the evolution of the partials of the variables (and partials of the functions) with respect to the initial conditions and with respect to to the parameters. The option to do that has four possible formats

- `AddPartials-> {{u,v,..}, s}`
- `AddPartials-> {{u,v,..}, s, Until}`
- `AddPartials-> {{u,v,..}, s, Only}`
- `AddPartials-> {{u,v,..}, listOfOrders}`

The list `{u,v,...}` represents the symbols of the elements with respect to we want the derivatives. The symbols of this list are symbols of the variables or symbols of the parameters. If the symbol corresponds to a variable the partials with respect to the initial value of this variables computed. If the symbol correspond to a parameter the partial with respect to the parameter is computed.

An integer `s` represents the total maximum order of the partials to compute.

If no third argument appear (or the third argument is the symbol `Until`) , all the partials until total order `s` are computed. If the third argument is the symbol `Only`, only the partial derivatives of total order `s` are computed.

If the second argument, `listOfOrders`, is a list, only the partials of the orders in the list are computed.

Let's assume a differential equation with three variables x, y, z and two parameters a, b. Then

- `AddPartials-> {{y,a}, 2}` computes

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

- `AddPartials-> {{y,a}, 2, Only}` computes

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

- `AddPartials-> {{y,a}, {{2,3},{1,2}}}` computes

$$\frac{\partial^5 x}{\partial y_0^2 \partial a^3}, \frac{\partial^3 x}{\partial y_0 \partial a^2}, \frac{\partial^5 y}{\partial y_0^2 \partial a^3}, \frac{\partial^3 y}{\partial y_0 \partial a^2}, \frac{\partial^5 z}{\partial y_0^2 \partial a^3}, \frac{\partial^3 z}{\partial y_0 \partial a^2}.$$

If a function G is added with the option `AddFunction`, the partials of this function with respect to the corresponding variables are added to the computation.

3.2.6 Options to change the output of the integrator in the driver

Option `Output`

This options declares where the solution (dense or not) is written. There are two posiblilities

```
Output -> Screen
Output -> "file"
```

In the first case the solution is written on the screen, in the second case into a file named `file`. By default no output is written.

In the minimal versions if the output is not sending into the screen the solution in `t0` and the solution in `tf` is written on the screen.

Option `DataMatrix`

Option only for standard versions. By default `DataMatrix->False`, but there are two other posibilities

```
DataMatrix -> True
DataMatrix -> "nameDM"
```

`DataMatrix` declares a bidimensional array where the solution is stored. The name is `nameDM` in the second case or the name of the file joined to `"_DataMatrix"` in the first case.

Each row corresponds to the solution in the point t_i of the integration interval. The first row represents the initial point. The last row represents the final point.

The number of columns is sufficient to store t_i, the variables in t_i, the functions in t_i, the partial derivatives of variables and functions in t_i.

3.2.7 Options to change the parameters of the TSM Integrator in the driver

The following options change the parameters of the numerical integrator. The default values are the best election for the most general cases and usually it is not necessary to change them

Options `Factor1`, `Factor2` and `Factor3`

`Factor1`, `Factor2` and `Factor3` change the parameters `fac1`, `fac2` and `fac3` respectively.

Option `MaxStepRatio` and `MinStepRatio`

`MaxStepRatio` and `MinStepRatio` change the parameters `rmaxstep` and `rminstep` respectively.

Option `MinOrder`

`MinOrder` changes the parameter `minord`.

Option `MaxIterationsNumber`

`MaxIterationsNumber` changes the parameter `nitermax`.

Option `OrderIncrement`

`OrderIncrement` changes the parameter `nordinc`.

Option `DefectErrorControl`

`DefectErrorControl` declares if the TSM Integrator uses the defect error control or not.

Chapter 4

How to use the Minimal versions of the **TSM Integrator**

Usually the drivers (main programs) generated with `CodeFiles` are sufficient to integrate one ODE. In this chapter we will learn how to run the minimal versions of the TSM Integrator, from the driver, and we will explain the driver and the prototypes of the main functions of the kernel of the TSM Integrator in order to change the driver or to write another different one.

As an example let's suppose again the differential equation that defines the sinus and cosinus functions

$$\frac{dx}{dt} = y, \quad \frac{dy}{dt} = -x, \quad x(0) = 0,\, y(0) = 1. \tag{4.1}$$

With MathTIDES we declare the differential equation `sincos`

In[9]:=

```
sincos = FirstOrderODE[{y, -x}, t, {x, y}];
```

4.1 **minf-tides** version

To integrate the equation (4.1) between 0 and 2π and write a dense output solution (intervals of length equal to 1.0) on the screen, we create the driver by writing, in MathTIDES, the expression

---

In[10]:=

```
CodeFiles[sincos, "sincosf", InitialConditions -> {0, 1},
     MinTIDES -> "Fortran", IntegrationPoints -> {0. , 2 Pi, Delta[1.]},
     Output ->  Screen]
```

Out[10]=

```
Files "dr_sincosf.f", "sincosf.f" and "minc_tides.f"
       written on directory ........
```

Finally write in your terminal the following command lines

```
$gfortran -O2 dr_sincosf.f sincosf.f minf_tides.f -o sincosf
$./sincosf
```

and you will obtain on the screen

```
   0.0000000000000000E+00   0.0000000000000000E+00   0.1000000000000000E+01
   0.1000000000000000E+01   0.8414709848078965E+00   0.5403023058681398E+00
   0.2000000000000000E+01   0.9092974268256817E+00  -0.4161468365471426E+00
   0.3000000000000000E+01   0.1411200080598671E+00  -0.9899924966004455E+00
   0.4000000000000000E+01  -0.7568024953079282E+00  -0.6536436208636121E+00
   0.5000000000000000E+01  -0.9589242746631387E+00   0.2836621854632264E+00
   0.6000000000000000E+01  -0.2794154981989259E+00   0.9601702866503660E+00
```

If you change the option `Output -> Screen` by the new one `Output -> "dataf"`, a file named `dataf` with the previous numbers is created an the program shows on the screen the values at the variables in the first and last points

```
 t =   0.0000000000000000E+00    X =   0.0000000000000000E+00   0.1000000000000000E+01
 t =   0.6283185307179586E+01    X =  -0.3330669073875470E-15   0.1000000000000000E+01
```

Let's note that the dense output (on screen or into a file) ends at the integration point 6.0 instead of the final point 2π, but the screen non-dense output writes the value at the end point of integration. The standard version works in a different way.

Now let's see the driver in order to understand how to change it or how to create a new one:

```
C-----------------------------------------------------------------------
C     Driver file of the MinF_TIDES program
C
C     This file has been created by MathTIDES. December 4, 2009, 13:08
C     Copyright (C) 2010 GME-Unizar
C     Authors: Abad, A., Barrio, R., Blesa, F. and Rodriguez, M.
C-----------------------------------------------------------------------
      Program  dr_sincosf
      IMPLICIT NONE
      INTEGER  i,j
C --- NUMBER OF VARIABLES AND PARAMETERS
      INTEGER  NVAR,NPAR
      PARAMETER  (NVAR = 2)
      PARAMETER  (NPAR = 1)
C --- TOLERANCES
      REAL*8 tolabs,tolrel
C --- TIMES: INITIAL, FINAL, INCREMENT
      REAL*8 tini, tend, dt
C --- VARIABLES AND PARAMETERS
      REAL*8 v(NVAR)
      REAL*8 p(NPAR)
C --- FILE NAME AND UNIT NUMBER OF DENSE OUTPUT
      CHARACTER fname*20
      INTEGER   FL
C --- OPTIONS
      LOGICAL dense_output, defect_error_control
C --- COUNTERS
      INTEGER accepted_steps, rejected_steps
C --- CONSTANTS OF THE METHOD (safety factors, maximum order, ...)
      REAL*8 fac1,fac2,fac3,rminstep,rmaxstep
      INTEGER nitermax,nordinc,minord,maxord
C --- GLOBALS
      COMMON /OPT/ dense_output, defect_error_control
      COMMON /ARS/ accepted_steps, rejected_steps
      COMMON /CONSTMET1/ fac1,fac2,fac3,rminstep,rmaxstep
      COMMON /CONSTMET2/ nitermax,nordinc,minord,maxord
      COMMON /FILE/ FL

C-------------------------------------------------------------------------------
C-------------------------------------------------------------------------------
C     INITIAL CONDITIONS,  INTEGRATION TIMES, TOLERANCES
C     Change ***** by numerical values if it is necesary
C-------------------------------------------------------------------------------
C-------------------------------------------------------------------------------

C --- INITIAL VALUES
      v(1) = 0d0
      v(2) = 0.1d1

C --- INITIAL INTEGRATION POINT
      tini = 0.d0

C --- ENDPOINT OF INTEGRATION
      tend = 6.283185307179586d0

C --- DELTA t FOR DENSE OUTPUT
      dt   = 1.d0

C --- REQUIRED TOLERANCES
      tolrel = 1.d-16
      tolabs = 1.d-16

C-------------------------------------------------------------------------------
C-------------------------------------------------------------------------------
C       DENSE OUTPUT (file , screen or none)
C-------------------------------------------------------------------------------
C-------------------------------------------------------------------------------

      FL = 6


C-------------------------------------------------------------------------------
C-------------------------------------------------------------------------------
C       CALL THE INTEGRATOR
```

```
75  C-----------------------------------------------------------------------------
76  C-----------------------------------------------------------------------------
77
78        CALL minf_tides(v,NVAR,p,NPAR,tini,tend,dt,
79       &   tolrel,tolabs,fname)
80
81        STOP
82        END
```

From lines 9 to 37 all the variables needed on the integration process are declared. There are parameters needful on the driver to call the integration routine `minf_tides`, and common parameters that contains the default options of the integration method.

From line 46 to line 61 the value of the variables, parameters, interval of integration and tolerances are declared. All of them are arguments of the integration routine `minf_tides`.

Line 69 contains a definition to the output unit, in this case the screen. If you change the option `Output -> Screen` by the new one `Output -> "dataf"` the lines 69-70 are substituted by

```
      FL = 72
      OPEN (UNIT = FL, FILE = 'dataf', STATUS = 'UNKNOWN')
```

The integration with minf-tides is made by calling the following subroutine

```
      SUBROUTINE minf_tides(v,numvar,p,numpar,tini,tend,dt, tolrel,tolabs)

      INTEGER numvar,numpar
      REAL*8  v(numvar),p(numpar)
      REAL*8  tini,tend,dt,tolrel,tolabs
```

where the input arguments are the following

1. `v` is a real vector of dimension `numvar`. On input it contains the initial value of the variables (value of the variables in `tini`). On output it stores the value of the variables in the final time `tend`.
2. `numvar` is an integer value with the number of variables.
3. `p` is a real vector of dimension `numpar` (1 if `numpar` $= 0$) with the value of the parameters.
4. `numpar` is an integer value with the number of parameters.
5. `tini` is a real number with the initial integration point.
6. `tend` is a real number with the final integration point. It can be lesser or greater than `tini`
7. `dt` is a real (negative if `tend < tini`) number with the increment in time where the dense output is computed. The dense output is computed in: {`tini, tini+dt, tini+2*dt, ...tini+k*dt`}, the last point verifies `tini+k*dt <= tend <tini+(k+1)*dt`. Not always the last point of the dense output coincides with the end integration point `tend`.
8. `tolrel` is a real number with the relative tolerance.
9. `tolabs` is a real number with the absolute tolerance.

The only output of this routine is the value of the variables `v` in the final time `tend`.

4.2 minc-tides version

To integrate the equation (4.1) between 0 and 2π and write the a dense output solution (intervals of length equal to 1.) on the screen, we create the driver by writing, in MathTIDES the expression

In[11]:=

```
CodeFiles[sincos, "sincosc", InitialConditions -> {0, 1},
    MinTIDES -> "C", IntegrationPoints -> {0. , 2 Pi, Delta[1.]},
    Output -> Screen]
```

Out[11]=

```
Files "dr_sincosc.c", "sincosc.c" , "sincosc.h" , "minc_tides.c" and "minc_tides.h"
     written on directory ........
```

Finally write in your terminal the following command lines

```
$gcc -O2 dr_sincosc.c sincos.c minc_tides.c -o sincosc -lm
$./sincosc
```

and you will obtain on the screen the same solution that in the minf-tides version.

Like in the minf-tides version if you change the option `Output -> Screen` by the new one `Output -> "datac"`, a file named `datac` with the previous numbers is created an the program shows on the screen the values of the variables at the first and last points. Let's note that the dense output (on screen or into a file) ends at the integration point 6.0 instead of the final point 2π, but the screen non-dense output writes the value at the end point of integration. The standard version works in a different way.

Now let's see the driver in order to understand how to change it or how to create a new one:

```
/****************************************************************************
        Driver file of the MinC_TIDES program

        This file has been created by MathTIDES. December 4, 2009, 17:44
        TIDES. Copyright (C) 2010 GME-Unizar
        Authors: Abad, A., Barrio, R., Blesa,F. and Rodriguez, M.
*****************************************************************************/

#include "minc_tides.h"

int main() {

        int  i, VARS, PARS;
        VARS = 2;
        PARS = 1;
        double tolrel, tolabs, tini, tend, dt;
        double v[VARS], p[PARS];
        extern FILE     *fd;



22  /************************************************************/
23  /************************************************************/
24  /*      INITIAL CONDITIONS, INTEGRATION TIMES, TOLERANCES    */
25  /*      Change *****  by numerical values if it is necesary */
26  /************************************************************/
27  /************************************************************/
28
29  /* --- INITIAL VALUES --- */
30          v[0] = 0e0 ;
31          v[1] = 0.1e1 ;
32
33  /* --- INITIAL INTEGRATION POINT --- */
34          tini = 0.e0 ;
35
36  /* --- ENDPOINT OF INTEGRATION   --- */
37          tend = 6.283185307179586e0 ;
38
39  /* --- DELTA t FOR DENSE OUTPUT  --- */
40          dt   = 1.e0 ;
41
42  /* --- REQUIRED TOLERANCES --- */
43          tolrel = 1.e-16 ;
44          tolabs = 1.e-16 ;
45
46  /***********************************************************/
47  /***********************************************************/
48  /*             DENSE OUTPUT (file, screen or none)          */
49  /***********************************************************/
50  /***********************************************************/
51
52          fd = stdout;
53
54  /***********************************************************/
55  /***********************************************************/
56  /*       CALL THE INTEGRATOR                               */
57  /***********************************************************/
58  /***********************************************************/
59
60          minc_tides(v,VARS,p,PARS,tini,tend,dt,tolrel,tolabs);
61
62
63          return 0;
64  }
```

From lines 13 to 18 all the variables needed on the integration process are declared. There are parameters needful on the driver to call the integration routine `minc_tides`. The global parameters that contains the default options of the integration method are declared only when they are changed with options of `CodeFiles`.

From line 29 to line 44 the value of the variables, parameters, interval of integration and tolerances are declared. All of them are arguments of the integration routine `minc_tides`.

Line 52 contains a declaration of the output file, in this case the screen (`stdout`). If you change the option `Output -> Screen` by the new one `Output -> "datac"` the line 52 is substituted by

```
        fd = fopen("datac", "w");
```

The integration with minc-tides is made by calling the following function

```
void  minc_tides(double *var,  int nvar,  double *par,  int npar,
          double tini, double tend, double dt, double tol_rel, double tol_abs);
```

where the input arguments are the following

1. `v` is a pointer to an array of dimension `numvar`. On input the array contains the initial value of the variables (value of the variables in `tini`). On output it stores the value of the variables in the final time `tend`.
2. `numvar` is an integer value with the number of variables.
3. `p` is a pointer to an array of dimension `numpar` (1 if `numpar` = 0) with the value of the parameters.
4. `numpar` is an integer value with the number of parameters.
5. `tini` is a `double` with the initial integration point.
6. `tend` is a `double` with the final integration point. It can be lesser or greater than `tini` .
7. `dt` is a `double` (negative if `tend < tini`) with the increment in time where the dense output is computed. The dense output is computed in: {`tini, tini+dt, tini+2*dt, ...tini+k*dt`}, the last point verifies `tini+k*dt <= tend <tini+(k+1)*dt`. Not always the last point of the dense output coincides with the end integration point `tend`.
8. `tolrel` is a `double` with the relative tolerance.
9. `tolabs` is a `double` with the absolute tolerance.

The only output of this routine is the value of the variables `v` in the final time `tend`.

Chapter 5

The library **LibTIDES**. Standard versions of the **TSM Integrator.**

The kernel of the standard versions of TIDES is contained into the C-library LibTIDES. This library will be described in this chapter.

5.1 Compiling and running the code.

Before to describe LibTIDES let's remember that, to use it, we need to use previously MathTIDES in order to create the linked functions with the iterations needful to obtain the Taylor series. When we call `CodeFiles` we pass a string as a second argument. This string represents the name associated with all the files created by `CodeFiles`. Let's suppose, along this chapter, that we use the name `"intfunct"`, as the second argument, then it writes two files named `intfunct.c`, `intfunct.h` and, if we want, a third file with the driver named `dr_intfunct.c`.

The file `intfunct.c` contains a function named `intfunct` that contains the iterations needful to create the Taylor Series solution. This function represents the differential equation and it is passed as the first argument in every version of the integrator.

To compile and run this code, to integrate the differential equation, write on the command line:

```
#gcc -O2 dr_intfunct.c intfunct.c  -o intfunct  -lTIDES -lm
#./intfunct
```

for the double precision version, and

```
#gcc -O2 dr_intfunct.c intfunct.c  -o intfunct  -ltides -lmpfr -lgmp -lm
#./intfunct
```

for the multiple precision version. Let's note that to link the MPFR library we need to include the GMP library[1].

If you write your own driver don't forget to include the header file `intfunct.h` that include all the TIDES necessary headers to integrate the problem.

```
#include "intfunct.h"
```

[1]Following the MPFR instruction you must call `-lmpfr` before `-lgmp`.

5.2 The double precision integrator

The principal function of the double precision standard version of the integrator is the following:

```
void dp_tides(LinkedFunction fcn,
        int nvar, int npar, int nfun,
        double x[], double p[],
        double lt[], int ntes,
        double tolrel, double tolabs,
        double **mat, FILE* fileout);
```

The arguments of both functions are all equal except those arguments relative to the integration points.

- *The linked function:* `fcn` is a pointer to the function that contains the iterations needful to create the Taylor Series solution. It will we described in Section 5.4
- *The dimensions of the problem:* `nvar, npar`, are two integer numbers that represent, respectively, the number of variables. `nfun` must be zero in this version.
- *Initial value of the variables:* `x` is a pointer to a `double` that represents an array with `nvar` elements. On input it has the value of the initial conditions (value of the variables at the initial point). On output it has the value of the variables at the final integration point.
- *Value of the parameters:* `p` is a pointer to a `double`, or an array with `npar` elements. It has the value of the parameters.
- *Integration points:* The integration points are represented by two arguments `lt` and `ntes`. `lt` is a pointer to a `double` that represents an array of dimension `ntes` that contains the list $\{t_0, \ldots, t_k\}$ of points where the solution will be computed. These points can be non-equidistants. The list must be ordered, but the order can be crescent or decrescent (for backward integration).
- *Tolerances:* `tolrel, tolabs` are two `double` variables with the relative and absolute tolerance of the method.
- *Output of the integrator:* `mat` is a double pointer to a `double` that represent a data matrix where the output will be stored. `fileout` is a pointer to a `File` where the output will be written on. More details about these outputs appear on the Section 5.5.

5.3 The multiple precision integrator

To handle real numbers in the multiple precision version we substitute the type `double` by the type `mpfr_t`, defined in the MPFR library. The variables of this type must be declared and initialized before giving them numerical values.

When `CodeFiles` is used to create a multiple precision integrator the number of precision digits is declared. This fact permits to simplify the process of initialize and assign values of the `mpfr_t` by using modified[2] functions. For instance to use a variable named `tend` with the value of π, and 40 precision digits we write

```
mpfr_t  tend;
mpfrts_init(& tend);
mpfrts_set_str(&tend, "3.1415926535897932384626433832795028841 97");
```

[2]modified with respect to the standard use of `mpfr`

The values of the variables are assigned by means of string of the desired length.

When we use the mp-tides version it is important to know that MathTIDES tries to transform all the (real) numbers of the process to the required precision. However, sometimes, we can find real numbers initialized only with double precision that invalidates the multiple precision process. In order to avoid these problems we must take care in the construction of the ODE and follow the next rule: never use real numbers in the expression of the differential equation, use instead integers, rationals or symbols like `Pi, E`, etc. If you need to use a real number declare it as a parameter and give the value of the paramater in the driver with the required precision. The problems can appear too when you use the option `InitialPoint->t0, tf, Points[n]`. For instance `InitialPoint->0, Pi, Points[100]` works fine, but `InitialPoint->0., Pi, Points[100]` works bad. Use, as possible the same rule that before or inspect the drive and fix it.

The principal function of the multiple precision standard version of the integrator is the following:

```
void mp_tides(LinkedFunction fcn,
        int nvar, int npar, int nfun,
        mpfr_t x[], mpfr_t p[],
        mpfr_t lt[], int ntes,
        mpfr_t tolrel, mpfr_t tolabs,
        mpfr_t** mat, FILE* fileout);
```

The arguments of both functions are all equal except those arguments relative to the integration points.

- *The linked function:* `fcn` is a pointer to the function that contains the iterations needful to create the Taylor Series solution. It will we described in Section 5.4
- *The dimensions of the problem:* `nvar, npar`, are two integer numbers that represent, respectively, the number of variables. `nfun` must be zero in this version.
- *Initial value of the variables:* `x` is a pointer to a `mpfr_t`, or an array with `nvar` elements. On input it has the value of the initial conditions (value of the variables at the initial point). On output it has the value of the variables at the final integration point.
- *Value of the parameters:* `p` is a pointer to a `mpfr_t`, or an array with `npar` elements. It has the value of the parameters.
- *Integration points:* The integration points are represented by two arguments `lt` and `ntes`. `lt` is a pointer to a `mpfr_t` that represents an array of dimension `ntes` that contains the list $\{t_0, \ldots, t_k\}$ of points where the solution will be computed. These points can be non-equidistants. The list must be ordered, but the order can be crescent or decrescent (for backward integration).
- *Tolerances:* `tolrel, tolabs` are to `mpfr_t` variables with the relative and absolute tolerance of the method.
- *Output of the integrator:* `mat` is a double pointer to a `double` that represent a data matrix where the output will be stored. `fileout` is a pointer to a `File` where the output will be written on. More details about these outputs appear on the Section 5.5.

5.4 The linked function

The linked function that contains the iterations needful to create the Taylor Series solution is passed to the integrator by means to a pointer to a function whose prototype is one of the following

```
typedef long (*LinkedFunction)(double t, double v[],
                double p[], int orden, double cvfd[][orden+1]);

typedef long (*LinkedFunction)(mpfr_t t, mpfr_t v[],
                mpfr_t p[], int orden, mpfr_t cvfd[][orden+1]);
```

depending on if we use the double precision version or the multiple precision version.

Let's take the example showed in the Section 5.1 of this chapter in which the file `intfunct.c` has been created from the expression `CodeFiles`. This file contains a function named `intfunct` (the same name as the second argument of `CodeFiles`) that follows the rules of the `LinkedFunction` prototype. Then, the symbol `intfunct` can be directly used as the firs argument of the integrator.

Let's note that this system permits to use the TSM Integrator to integrate several ODEs inside the same main program (on the contrary that in the minimal versions). We only need to create with `CodeFiles` files with different names and call the integrator with these different functions.

5.5 The output of the integrator

The two last arguments of the integrator tell it what kind of output we want. There are two possibilities: a data matrix or a file (this file can be a physical file or the screen), both with the (same) solution.

The outputs have `nrows` rows or lines equal to the number $k+1$ of points $\{t_0, t_1, \ldots, t_k\}$ in which we compute the solution. Each row represents the solution on each point t_i.

Each row has `ncolumns` elements (columns of the output). This number depends on the options that we choose in `CodeFiles`, and can be obtained by calling the function

```
long intfunct_columns();
```

The name of this function is formed adding `_columns` to the string of the second argument of `CodeFiles`. It returns an integer number (`long`) with the value of `ncolumns`.

The two last arguments of the integrator are `double** mat` (or `mpfr_t** mat` in the multiple precision case) and `FILE* fileout`. If we don't want any of these outputs we will pass a value equal to `NULL` to the integrator. In other case we need to declare a variable of the corresponding type, give it a value and pass it to the integrator.

To write into a file you must include in your code the lines

```
FILE   *mydatafile;
mydatafile = fopen("datafilename", "w");
```

where `datafilename` will be the name of the file. If we want to write into the screen instead a file substitute this line by `mydatafile = stdout;`. After that use `mydatafile` as the last argument.

If you want to write into a data matrix you need to declare and initialize it as follows

```
double** mydatamatrix;
Array2DB_init(&mydatamatrix, nrows, ncolumns);
```

in the double precision case, and

```
mpfr_t** mydatamatrix;
Array2MP_init(&mydatamatrix, nrows, ncolumns);
```

in the multiple precision case.

The function `Array2DB_init` (`Array2MP_init`) allocate storage for a bidimensional array type `double` (`mpfr_t`) with `nrows` rows and `ncolumns` columns.

Eventually we need to understand what information is contained in each column of the output in order to use it. This may be difficult when we compute partial derivatives. In order to identify each column let's remain that the first file of the file correspond with the column of order zero of the data matrix. Then in what follows we identify each position with the index of the data matrix. The column of order zero represents the time t. In order to know in what column appears a particular partial derivative of a variable we may use the function

```
long intfunct_variable_column(int v, char *der);
```

The name of this function is formed adding `_variable_column` to the string of the second argument of `CodeFiles`. It returns the position of the partial derivative `der` of the variable `v`. The variable is represented by an integer `v` that represents the index of the variable following the C-style for indexes (0 is the first index). To identify each derivative we use its representation by means of indexes $\mathbf{i} = \{i_1, i_2, \dots i_m\}$. Each character of the string `der` coincides with each number i_k.

Let's suppose an example with three variables $\{x, y, z\}$ and several parameters σ, β. Let's suppose we compute the partials with respect to $\{x, y, z, \sigma\}$. Then the position of $\partial^2 y/\partial x_0 \partial\sigma$ is given by the function `lorenzP_variable_column(1, "1010")`. The variable x is at the position `lorenzP_variable_column(0, "0000")`, that returns the value 1.

5.6 Internal parameters of the integrator

In Section 2.2 of this document we discuss a set of internal parameters of the TSM Integrator. All this parameters, except to the tolerances, have a default value that usually, for the most general cases, are the best election, and the user does not need to change them. However, there are two ways to change this parameters: by options of `CodeFiles` that change the driver (see 3.2.7), or by changing directly the value if we write our own driver.

The way to access to the parameters is by declaring the parameters as external variables in the main program (driver)

```
extern double   fac1;
extern double   fac2;
extern double   fac3;
extern double   rmaxstep;
extern double   rminstep;
extern int      nitermax;
extern int      nirdinc;
extern int      minord;
extern int      defect_error_control;
```

where the meaning of names are the same that in 2.2.

The default values of the parameters are

```
fac1 = 0.95;
fac2 = 10.;
fac3 = 0.8
rmaxstep = 1e2;
rminstep = 1e-2;
nitermax = 5;
nirdinc = 4;
minord = 6;
defect_error_control = 0   /* = False */
```

To change the value of any of them include a similar line in the driver. For instance to use the defect error control write

```
defect_error_control = 1;
```

5.7 Two examples of driver

Let's take again the differential equation (3.2) that define the sinus and cosinus functions.

In[12]:=

```
sincos = FirstOrderODE[{y, -x}, t, {x, y}]
```

First example

In the first case let's compute, in double precision the solution, in $[0, 2\pi]$, together with the partial derivatives with respect to the initial conditions of second order, i.e. $\partial^2 x/\partial x_0^2$, $\partial^2 x/\partial y_0^2$, $\partial^2 x/\partial x \partial y$, and the same for y.

To write the driver file `dr_sincosp.c` we write in MathTIDES

In[13]:=

```
CodeFiles[sincos, "sincosp",
    InitialConditions -> {0, 1},
    IntegrationPoints -> {0, Pi/2, Pi, 2 Pi},
    DataMatrix -> "scdata",
    DefectErrorControl -> True,
    AddPartials -> {{x, y}, 2, Only}]
```

Out[13]=

```
Files "dr_sincosp.c", "sincosp.h" and sincosp.c" written on
    directory ....."
```

and finally the file `dr_sincosp.c` is

```
#include <stdio.h>
#include <stdlib.h>
#include "sincosp.h"

int main() {

        int nvar = 2;
        int npar = 0;
        int nfun = 0;
        int nipt = 4;
        double v[nvar], lt[nipt];
        double tolrel, tolabs;
        double** scdata;
        FILE   *fd;

/************************************************************/
/************************************************************/
/*       CONSTANTS OF THE METHOD                            */
/************************************************************/
/************************************************************/

        extern int defect_error_control;
        defect_error_control = 1;

/*************************************************************/
/*************************************************************/
/*     INITIAL CONDITIONS, INTEGRATION TIMES, TOLERANCES    */
/*     Change *****  by numerical values if it is necesary  */
/*************************************************************/
/*************************************************************/

/* --- INITIAL VALUES --- */
        v[0] = 0 ;
        v[1] = 1. ;

/* ---     INTEGRATION POINTS    --- */
        lt[0] = 0. ;
        lt[1] = 1.570796326794897 ;
        lt[2] = 3.141592653589793 ;
        lt[3] = 6.283185307179586 ;

/* --- REQUIRED TOLERANCES --- */
        tolrel = 1.e-16 ;
        tolabs = 1.e-16 ;

/************************************************************/
/************************************************************/
/*        OUTPUT:         data matrix                      */
49  /********************************************************/
50  /********************************************************/
51
52          Array2DB_init(&scdata, nipt, sincosp_columns());
53
54
55  /********************************************************/
56  /********************************************************/
57  /*       CALL THE INTEGRATOR                            */
58  /********************************************************/
59  /********************************************************/
60
61          dp_tides(sincosp, nvar, npar, nfun, v, NULL,
62                          lt, nipt, tolrel, tolabs, scdata, NULL);
63
64
65          return 0;
66  }
67
```

Lines 1 to 3 include the header files of the system and the header file created with `CodeFiles`.

Lines 7-9 declare the number of variables and functions and, in the appropriate cases, the number of parameters. Line 10 declares the number of points in which the output is written (the number of integration points plus the initial point). Lines 11-14 declare the local variables of the driver, all of them necessaries to call the integrator.

Lines 22-23 declare, as external, and change the values of the internal parameters of the method that we change on the driver. In this example we use defect error control in the integration. If we do not change any parameter of the method these lines that not appear.

Lines 33-44 assign value to the initial variables, integration points and tolerances.

Line 52 initialize the data matrix to store the solution. It has 4 rows ($t_0 = 0, t_1 = \pi/2, t_2 = \pi, t_3 = 2\pi$). and the number of columns is given by the function `sincosp_columns()`.

Lines 61-62 call the integrator. A `NULL` is used in the pointer to the parameters (no parameter in the problem) and in the output file (no output file, only data matrix).

In the data matrix, the integration time appears in the first column. Columns 2 and 3 contains the values $x(t_i), y(t_i)$. To know in what column the partials appear use the function `sincosp_variable_column`. For instance the partial $\partial^2 x/\partial x_o^2$ is in column `sincosp_variable_column(0,"20")`, and $\partial^2 y/\partial x_o \partial y_0$ appears in column `sincosp_variable_column(1,"11")`.

Second example

Now we will compute only the sinus and cosinus, but we will make the calculation with 40 precision digits. The output has 100 integration points (101 including the initial point), from 0 to 2π, it will appear on the screen and in a data matrix.

In[14]:=

```
CodeFiles[sincos, "sincosmp",
     InitialConditions -> {0, 1},
     IntegrationPoints -> {0, 2 Pi, Points[100]},
     DataMatrix -> "mpscdata",
     Output -> Screen,
     PrecisionDigits -> 40];
```

Out[14]=

```
Files "dr_sincosmp.c", "sincosmp.h" and sincosmp.c" written on \
directory  .....
```

The driver file `dr_sincosmp.c` is

```
#include <stdio.h>
#include <stdlib.h>
#include "mpfr.h"
#include "sincosmp.h"

int main() {

        set_precision_digits(40);

        int i;
        int nvar = 2;
        int npar = 0;
        int nfun = 0;
        int nipt = 101;
        mpfr_t v[nvar], lt[nipt];
        mpfr_t tolrel, tolabs;
        mpfr_t** mpscdata;
        FILE   *fd;


/************************************************************/
/************************************************************/
/*     INITIAL CONDITIONS, INTEGRATION TIMES, TOLERANCES    */
/*     Change *****  by numerical values if it is necesary */
/************************************************************/
/************************************************************/

/* --- INITIAL VALUES --- */
        for(i=0; i<nvar; i++) mpfrts_init(&v[i]);
        mpfrts_set_str(&v[0], "0");
        mpfrts_set_str(&v[1], "1.");

/* ---     INTEGRATION POINTS    --- */
        for(i=0; i<nipt; i++) mpfrts_init(&lt[i]);
        mpfr_t t0,dt,idt;
        mpfrts_init(&t0);
        mpfrts_init(&dt);
        mpfrts_init(&idt);
        mpfrts_set_str(&t0, "0");
40          mpfrts_set_str(&dt, "0.06283185307179586476925286766559005768394");
41          for(i=0; i<nipt; i++) {
42                  mpfrts_mul_i(&idt,dt,i);
43                  mpfrts_add(&lt[i],t0,idt);
44          }
45
46  /* --- REQUIRED TOLERANCES --- */
47          mpfrts_init(&tolrel);
48          mpfrts_init(&tolabs);
49          mpfrts_set_str(&tolrel, "1.e-40");
50          mpfrts_set_str(&tolabs, "1.e-40");
51
52  /***********************************************************/
53  /***********************************************************/
54  /*        OUTPUT: screen &data matrix                     */
55  /***********************************************************/
56  /***********************************************************/
57
58          fd = stdout;
59          Array2MP_init(&mpscdata, nipt, sincosmp_columns());
60
61  /***********************************************************/
62  /***********************************************************/
63  /*       CALL THE INTEGRATOR                               */
64  /***********************************************************/
65  /***********************************************************/
66
67          mp_tides(sincosmp, nvar, npar, nfun, v, NULL,
68                          lt, nipt, tolrel, tolabs, mpscdata, fd);
69
70
71          return 0;
72  }
```

In line 3 we include the file `mpfr.h` to work with the MPFR library.

Lines 30 to 32 declare the variables as `mpfr_t` type.

Lines 29 to 31 contains the initialization and assignation of values of the initial conditions, the integration points and lines 47-50 the tolerances. Let's note that the string with the values of the variables has been computed in Mathematica with the adequate precision.

Lines 34-44 contain the creation of the list `lt` of integration points.

Finally, the output file now is the screen (line 58).

Bibliography

[1] Abad, A., Barrio, R., Blesa, F. Rodriguez, M. 2010. TIDES: a Taylor series Integrator of Differential EquationS. Preprint (2010).

[2] Abad, A., Barrio, R., Blesa, F. Rodriguez, M. 2010. Symbolic Approach to the Numerical Taylor Series Method. Preprint (2010) .

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

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

[5] Barrio, R., 2006. Sensitivity analysis of ODE's/DAE's using the Taylor series method. SIAM J. Sci. Comput. 27 (6) 1929–1947.