Part III

TIDES reference guide

Chapter 12

MathTIDES reference guide

12.1 Representing ODEs in MathTIDES

The Taylor Series Method integrates only first order ODE systems. However, a higher order ODE, with certain conditions, may be transformed into a first order ODE system, a dynamical system described by a potential function V leads to a first order ODE system ($\dot{\boldsymbol{y}} = \boldsymbol{Y}, \dot{\boldsymbol{Y}} = \boldsymbol{F} = -\nabla V$), and the Hamilton's equations obtained from a Hamiltonian $\mathcal{H}$ are a first order ODE system.

In MathTIDES a first order ODE is represented by means of 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.

12.1.1 First order differential equations

A first order ODE is represented by the equation

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}(t, \boldsymbol{y}(t); \boldsymbol{p}), \quad \boldsymbol{y}(t_0) = \boldsymbol{y}_0, \quad \boldsymbol{y} \in \mathbb{R}^n, \quad \boldsymbol{p} \in \mathbb{R}^m, \tag{12.1}$$

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:

- *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 explicitly, or not, in the first argument.
- *Third argument:* the list $\{y_1, \ldots, y_n\}$ of symbols that represents 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 represents 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

$$\frac{dx}{dt} = y, \quad \frac{dy}{dt} = -x. \tag{12.2}$$

To declare this ODE we will write the expression

In[33]:=

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

Out[33]=

```
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

$$\frac{dx}{dt} = \frac{1}{\sqrt{1 - k^2 \sin^2 t}}, \tag{12.3}$$

that we declare with the expression

In[34]:=

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

Out[34]=

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

where the modulus k acts as a parameter.

12.1.2 Higher order differential equations

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

$$\boldsymbol{F}(t, \boldsymbol{y}, \frac{d\boldsymbol{y}}{dt}, \frac{d^2\boldsymbol{y}}{dt^2}, \ldots, \frac{d^k\boldsymbol{y}}{dt^k}; \boldsymbol{p}) = 0, \qquad \boldsymbol{y}(t_0) = \boldsymbol{y}_0, \ldots, \frac{d^k\boldsymbol{y}}{dt^k}(0) = \boldsymbol{y}_0^{(k)}, \tag{12.4}$$

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

Let us suppose that all the derivatives $y_1^{(k)}, \ldots y_n^{(k)}$ of the greatest order k appear explicitely in (12.4) then, solving the system (12.4) in $y_1^{(k)}, \ldots y_n^{(k)}$, if it is possible, we 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 `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.
  - The derivatives of greater order of all the variables must appear in the system.
- *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 represents 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 represents 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 has the symbol `x`, the derivatives of this variable are converted into new variables whose symbol begins by `x` 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 equations 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

$$\frac{d^2x}{dt^2} + \omega x = 0. \tag{12.5}$$

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

In[35]:=

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

Out[35]=

```
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 is the following third order ODE

$$x''' - 2y'' + x' = 2x^2 - y,$$
$$4y''' - 2x''y' = 2x + y^2.$$

In MathTIDES we will write

In[36]:=

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

Out[36]=

```
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.

12.1.3 From potential to first order ODEs

Let's suppose a potential $V(\boldsymbol{y}, \boldsymbol{p})$ in the variables $\boldsymbol{y} \in \mathbb{R}^n$, with m parameters $\boldsymbol{p} \in \mathbb{R}^m$, then the equation $\ddot{\boldsymbol{y}} = -\nabla V(\boldsymbol{y}, \boldsymbol{p})$ will be obtained as a first order ODE by means of the MathTIDES expression of head `PotentialToODE` that has 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 represents the variables. If $n = 1$ the argument is not a list.
- *Fourth argument:* the list $\{p_1, \ldots, p_m\}$ of symbols that represents 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

$$V = \frac{\mu}{\sqrt{x^2 + y^2 + z^2}}, \tag{12.6}$$

where μ represents a parameter.

In[37]:=

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

Out[37]=

```
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 equations into a first order equations duplicating the number of variables {`x, y, z, x$d1, y$d1, z$d1`}. The symbols of the new variables (derivatives) are formed by adding `$d1` to the symbol of the duplicate variables.

12.1.4 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

$$\frac{d\boldsymbol{x}}{dt} = \frac{\partial \mathcal{H}}{\partial \boldsymbol{X}}, \quad \frac{d\boldsymbol{X}}{dt} = -\frac{\partial \mathcal{H}}{\partial \boldsymbol{x}}. \tag{12.7}$$

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

- *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 represents 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 represents 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

$$\mathcal{H} = \frac{X^2 + Y^2}{2} - \frac{\mu}{\sqrt{x^2 + y^2}}, \tag{12.8}$$

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

In[38]:=

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

Out[38]=

```
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}]
```

12.2 Creating the TSM Integrator with MathTIDES

12.2.1 How to create the TSM Integrator

To write the C or FORTRAN code to use together with the TIDES library we will use an expression with head `TSMCodeFiles` 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:* optional arguments described later in this chapter.

12.2.2 Files written with `TSMCodeFiles`

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

- Minimal Version in C (minc-tides)
  - A driver (main program) named `"dr_name.c"`.
  - Two files `"name.c", "name.h"` with the differential equation (ODE files).
  - Compiling and running the previous files with the file `"minc_tides.c"` (written with option `TIDESFiles`), or linking them with the library LibTIDES we create the executable program.
- Minimal Version in FORTRAN (minf-tides)
  - A driver (main program) named `"dr_name.f"`.
  - A file `"name.f"` with the differential equation.
  - Compiling and running the previous files with the file `"minf_tides.f"` (written with option `TIDESFiles`), or linking them with the library LibTIDES we create the executable program.
- 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 (including **libmpfr.a** and **libgmp.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. 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[39]:=

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

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

12.2.3 Options to change the version of the integrator and the files written by TIDES

12.2.3.21 *Option*: `Driver`

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

12.2.3.22 *Option*: `ODEFiles`

With the option `ODEFiles -> False`, MathTIDES does not write the ODE files. The default is `True`. This option is useful after we create an integrator and we want to change only the driver.

12.2.3.23 *Option*: `MinTIDES`

`MinTIDES` is used to create files to use with the minimum versions of TIDES.
`MinTIDES -> "C"` creates the C minimum version minc-tides.
`MinTIDES -> "Fortran"` creates the FORTRAN minimum version minf-tides.
The default option, `MinTIDES -> False`, creates the standard version.

12.2.3.24 *Option*: `Precision`

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 `Precision`. By default this option has the value `Precision->Double`. This means that the standard double precision version dp-tides is created.

With the options `Precision->Multiple` or `Precision->Multiple[n]` a multiple precision version mp-tides is created. In the second case the integer `n` declares the number of precision digits to use in the integration.

If we want only the ODE files, and we do not want the driver, it is sufficient to use the option `Precision->Multiple` because these files work independently of the default precision that must be declared on the driver. When we create a driver we need the option `Precision->Multiple[n]`, where the integer `n` is the number of precision digits declared on the driver.

12.2.3.25 *Option*: `TIDESFiles`

With the option `TIDESFiles -> True` one of the files `minc_tides.c`, `minf_tides.f`, `dp_tides.h` or `mp_tides.h` (depending on the version) is written.

12.2.4 Options to change how to call the integrator

The following options only changes the driver and they do not affect to the ODE files.

12.2.4.26 *Option*: `InitialConditions`

With the option `InitialConditions -> { ...}` 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 numerical values, appear on the driver.

12.2.4.27 *Option*: `ParametersValue`

With the option `ParametersValue -> {...}` 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 numerical values, appear on the driver.

12.2.4.28 *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, 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.
  - With this option the solution is computed in $\{t_0, t_1, \ldots, t_k\}$ = {`t0, t0+dt, t0+2*dt, ..., t0+k*dt`}.
- `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`.
  - With this option the solution is computed in $\{t_0, t_1, \ldots, t_k\}$ = {`t0, t0+dt, t0+2*dt, ..., t0+k*dt = tf`}.
- `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.
- With this option 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, 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.
  - With this option the 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 (increasing or decreasing). They can be non-equidistant points.

12.2.4.29 *Option*: `RelativeTolerance, 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, for both tolerances, is $10^{(1-p)}$, where p is the number of precision digits declared with the option `Precision -> Multiple[p]` (10^{-16} for double precision integration). If only one tolerance is declared both are taken equal.

12.2.5 Option to compute the value of extra functions along the solution

12.2.5.30 *Option*: `AddFunctions`

This option changes the ODE files and the driver.

The integration of the system (1.1) gives the function $\boldsymbol{y}(t)$, i.e. the evolution over the time of the variables. Sometimes, we are interested in the evolution, along the solution of the system, of a dynamical variable defined by a function $G(t, \boldsymbol{y}, \boldsymbol{p})$, i.e. the function $G(t) = G(t, \boldsymbol{y}(t), \boldsymbol{p})$. Writing the option `AddFunctions-> {G1, G2,...}` we redefine

the differential equation to extend the application of the Taylor method to find the time evolution of the functions G1,G2, ...

For instance, let's suppose we want to check the value of the tangent, together with the sine and cosine in the system (6.1), then

In[40]:=

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

In[41]:=

```
TSMCodeFiles[sincos, "sincosf", AddFunctions -> {x/y} ];
```

In the example of the hamiltonian of the planar keplerian problem we may check how the energy maintains its value over the time, to do that we check the evolution of the Hamiltonian

In[42]:=

```
kepHam = (v^2 /2 - mu/r)/.{r -> Sqrt[x^2 + y^2], v -> Sqrt[X^2 + Y^2]};
```

In[43]:=

```
hamkepener = HamiltonianToODE[kepHam, t, {x, y, X, Y}, mu];
```

In[44]:=

```
TSMCodeFiles[hamkepener, "hamkepener",  AddFunctions -> {kepHam}];
```

12.2.6 Option to compute partial derivatives

12.2.6.31 *Option*: `AddPartials`

This option changes only the driver.

Together with the time evolution of the variables and functions we may compute the evolution of the partial derivatives of the variables (and partial derivatives of the functions) with respect to the initial conditions and with respect 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 compute 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 partial derivatives with respect to the initial value of this variable is computed. If the symbol corresponds to a parameter the partial with respect to the parameter is computed.

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

If no third argument appears (or the third argument is the symbol `Until`) , all the partial derivatives 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 partial derivatives of the orders in the list are computed. Then, supposing an ODE in which one of the variables has the symbol `y`, and one of the parameters has the symbol `a`,

- `AddPartials-> {{y,a}, 2}` or `AddPartials-> {{y,a}, 2}, Until}` compute
  $\frac{\partial}{\partial y_0}, \quad \frac{\partial}{\partial a}, \quad \frac{\partial^2}{\partial y_0^2}, \quad \frac{\partial^2}{\partial y_0 \partial a}, \quad \frac{\partial^2}{\partial a^2}.$
- `AddPartials-> {{y,a}, 2, Only}` computes $\frac{\partial^2}{\partial y_0^2}, \quad \frac{\partial^2}{\partial y_0 \partial a}, \quad \frac{\partial^2}{\partial a^2}.$
- `AddPartials-> {{y,a}, {{1,2},{2,3}}}` computes $\frac{\partial^3}{\partial y_0 \partial a^2}, \quad \frac{\partial^5}{\partial y_0^2 \partial a^3}.$

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

12.2.7 Options to change the output of the integrator

The following options only changes the driver and they do not affect to the ODE files.

12.2.7.32 *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 to the screen, the solution in `t0` and the solution in `tf` are written on the screen.

12.2.7.33 *Option*: `DataMatrix`

This option only works 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. In the first case the name of the data matrix is the name of the file joined to `"_DataMatrix"`, or `"_EventsVector"` when the option is used to compute events. In the second case the name is `nameDM`.

The dimensions of the matrix are declared inside the LibTIDES Taylor integrator. The number of rows corresponds to the number of points where the solution is computed (including the initial point as the first row), or the number of found events when the option is used to compute events. The number of columns must be sufficient to store, in this order

- The point t_i.
- The value of the variables in t_i: $\boldsymbol{x}(t_i)$.
- The value of the functions $G_i(t_i, \boldsymbol{x}(t_i), \boldsymbol{p})$ if `AddFunction` is used or the event function when the option is used to compute events.
- The value of the partial derivatives derivatives if they are computed.

12.2.8 Options to compute events

This options changes both, the driver and the ODE files, because, to compute events, it is necessary to compute an extra function.

12.2.8.34 *Option*: `FindZeros`

MathTIDES writes, with the option `FindZero->G`, the code to compute the zeros of $G(\boldsymbol{y}(t))$ inside an interval. `G` is the MATHEMATICA expression of the function $G(\boldsymbol{y})$.

12.2.8.35 *Option*: `FindExtrema`

MathTIDES writes, with the option `FindExtrema->G`, the code to compute the local extrema (maxima and minima) of $G(\boldsymbol{y}(t))$ inside an interval. `G` is the MATHEMATICA expression of the function $G(\boldsymbol{y})$.

12.2.8.36 *Option*: `FindMinima`

MathTIDES writes, with the option `FindMinima->G`, the code to compute the local minima of $G(\boldsymbol{y}(t))$ inside an interval. `G` is the MATHEMATICA expression of the function $G(\boldsymbol{y})$.

12.2.8.37 *Option*: `FindMaxima`

MathTIDES writes, with the option `FindMaxima->G`, the code to compute the local maxima of $G(\boldsymbol{y}(t))$ inside an interval. `G` is the MATHEMATICA expression of the function $G(\boldsymbol{y})$.

12.2.8.38 *Option*: `EventTolerance`

With the option `EventTolerance->...` we declare the tolerance of the numerical method used to find the zeros of a polynomial (a number is a zero if its absolute value is less than the tolerance). The default value is 10^{-16} if double precision is used, or 10^{-p}, where p is the number of precision digits declared with the option `Precision -> Multiple[p]`.

12.2.8.39 *Option*: `EventsNumber`

With the option `EventsNumber->...` we declare the maximum number of events that we want to compute inside the integration interval. Sometimes there are less events than this maximum number. The default options is `EventsNumber->0` that computes all the events inside the interval. When TIDES finds all the desired events before to reach the final integration point, the integration stops.

12.2.9 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. They only changes the driver and they do not affect to the ODE files.

12.2.9.40 *Option*: `Factor1, Factor2, Factor3`

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

12.2.9.41 *Option*: `MaxStepRatio, MinStepRatio`

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

12.2.9.42 *Option*: `MinOrder`

`MinOrder->...` changes the parameter `minord`.

12.2.9.43 *Option*: `OrderIncrement`

`OrderIncrement->...` changes the parameter `nordinc`.

12.2.9.44 *Option*: `DefectErrorControl`

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

12.2.9.45 *Option*: `MaxIterationsNumber`

`MaxIterationsNumber->...` changes the parameter `nitermax`. This parameter is used when the option `DefectErrorControl` is declared. If the TSM Integrator reaches the maximum iteration number without achieving the `DefectErrorControl` condition, it shows an error message.

12.2.9.46 *Option*: `KahanSummation`

`KahanSummation->...` declares if the TSM Integrator uses the `Kahan summation` when computes the central point of the series expansion. By default its value is `True`. This options works only with the standard version.

12.2.9.47 *Option*: `CompensatedHorner`

`CompensatedHorner->...` declares if the TSM Integrator uses the `Compensated Horner` algorithm to evaluate the Taylor series. By default its value is `False`. This options works only with the standard version.

12.3 MathTIDES function `PartialDerivativesText`

Another way to construct the driver to compute partial derivatives is by creating the integrator without using the option `AddPartials`, and change the driver manually (see 10.4). To do that we need to use the MathTIDES function `PartialDerivativesText`, that has four arguments:

1. A list with the symbols of the variables.

2. A list with the symbols of the parameters.

3. The third argument is equal to the expression after the arrow of the option `AddPartials`

4. An string that contains a name to construct the name of the array used in the driver.

The output is the text of the initialization of the array to compute partial derivatives. Copy and paste this text into the driver, and declare the array, and the partial derivatives will be computed.

Chapter 13

LibTIDES reference guide

13.1 LibTIDES and MPFR library

LibTIDES handles multiple–precision by using the MPFR library. There is not necessary to know MPFR if one uses the driver created by MathTIDES without modification, but, when one tries to understand the driver or one wants to change it, it is useful to read the user manual of MPFR and learn how TIDES uses MPFR. Let us begin by several basic ideas about MPFR.

- In MPFR the basic data type is `mpfr_t`. It represents a real number with the desired binary precision digits.
- Every `mpfr_t` variable must be initialized by using the function `mpfr_init2(var, prec)`, where `var` represents the variable to initialize and `prec` represents its precision (in bits).
- The precision of each `mpfr_t` variable represents the number of bits used when the variable is stored. By default precision is 53 bits (the number of bits used for a `double`).
- When MPFR makes any operation with a `mpfr_t` variable the way in which the result is rounded must be declared. The way to declare the rounding mode is by passing to the function that makes the operation one of the following arguments: `MPFR_RNDN, MPFR_RNDZ, MPFR_RNDU, MPFR_RNDD` (or `GMP_RNDN, GMP_RNDZ, GMP_RNDU, GMP_RNDD` with a version of the MPFR library previous to the version 3.0).
- The way in which the driver created by MathTIDES gives value to the `mpfr_t` variables is by using the function: `mpfr_set_str(var, str, b, rnd)`. After calling

this function the variable `var` takes the value represented by the string `str` in base `b`, and rounded in the way represented by `rnd`.

The precision and the rounding mode can be changed in MPFR for any variable and any operation. However in TIDES we define a working precision and rounding mode and we make all the operations and store every variable with the same precision and rounding mode.

In TIDES we declare decimal precision instead of binary precision. The function

```
void set_precision_digits(int dprec)
```

declares that every `mpfr_t` variable used in TIDES is stored with `dprec` decimals of precision. This function computes the number of necessary bits to work with this decimal precision and store it in the global variable `TIDES_PREC`, that is the second argument used any time that `mpfr_init2` is called. `TIDES_PREC` has a default value of 53 when `set_precision_digits()` is not used. It means that TIDES works with MPFR but in double precision (about 16 decimal digits).

The working rounding mode in TIDES is stored in the global variable `TIDES_RND`. Its default value is `MPFR_RNDN` (`GMP_RNDN` when a version of MPFR previous to the version 3.0 is used). To change the value of the working rounding mode use the function

```
void set_rounding_mode(mpfr_rnd_t rnd)
```

where `rnd` is one of the MPFR rounding modes.

13.2 LibTIDES functions to call the integrator

There are two LibTIDES functions to call the TSM Integrator in double precision.

```
void dp_tides_delta(DBLinkedFunction fcn,
        int *pdd,
        int nvar, int npar, int nfun,
        double *x, double *p,
        double t0, double dt, int nipt,
        double tolrel, double tolabs,
        dp_data_matrix *dmat, FILE* fileout);
```

```
void dp_tides_list(DBLinkedFunction fcn,
        int *pdd,
        int nvar, int npar, int nfun,
        double *x, double *p,
        double *lt, int ntot,
        double tolrel, double tolabs,
        dp_data_matrix *dmat, FILE* fileout) ;
```

and two more in multiple precsion

```
void mp_tides_delta(MPLinkedFunction fcn,
        int *pdd,
        int nvar, int npar, int nfun,
        mpfr_t x[], mpfr_t p[],
        mpfr_t tini, mpfr_t dt, int nipt,
        mpfr_t tolrel, mpfr_t tolabs,
        mp_data_matrix *dmat, FILE* fileout);


void mp_tides_list(MPLinkedFunction fcn,
        int *pdd,
        int nvar, int npar, int nfun,
        mpfr_t x[], mpfr_t p[],
        mpfr_t lt[], int ntot,
        mpfr_t tolrel, mpfr_t tolabs,
        mp_data_matrix *dmat, FILE* fileout);
```

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

- *The linked function:* `fcn` is a pointer to the function that contains the ODE function. In this argument we write the name used in the second argument of `TSMCodeFiles`.
- *The partial derivatives information:* `pdd` is a pointer to an integer that represents an array with the necessary information to compute the desired partial derivatives (see chapter 10). Use `NULL` when no partial derivative needs to be computed.

- *The dimensions of the problem:* `nvar, npar, nfun` are three integer numbers that represent, respectively, the number of variables, the number of parameters and the number of extra functions to evaluate.

- *Initial value of the variables:* `x` is a pointer to a `double` (`mpfr_t`) 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` (`mpfr_t`), or an array with `npar` elements. It has the value of the parameters. If there is no parameter this argument will be `NULL`.

- *Integration points* (cases `dp_tides_delta`, `mp_tides_delta`): the integration points are represented by three arguments: two `double` (`mpfr_t`) variables `tini, dt`, that contain the initial point and the increment and a `int` variable `nipt` with the number of equidistant points where we compute the solution (without including the initial point).

- *Integration points* (cases `dp_tides_list`, `mp_tides_list`): the integration points are represented by two arguments `lt` and `ntot`. `lt` is a pointer to a `double` (`mpfr_t`) that represents an array of dimension `ntot` 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 increasing or decreasing (for backward integration).

- *Tolerances:* `tolrel, tolabs` are two `double` (`mpfr_t`) variables with the relative and absolute tolerance of the method.

- *Output of the integrator:* `dmat` is a pointer to a `dp_data_matrix` (or `mp_data_matrix`) type 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.

13.3 Computing the position of each element of the output

Usually, when we compute partial derivatives, it is very difficult to know the position, at the output, of a particular element. The next two LibTIDES functions returns an integer number with the position of a particular partial derivative at the output (screen, file or data matrix). This position is zero based (0 means the first column, the time, i means the $(i+1)$-th column). They return -1 if the corresponding derivative is not computed. The value 0 is never returned because it corresponds to the column of the time t.

```
long position_variable(int v, char* der, int nvar, int nfun, int *pdd);
long position_function(int f, char* der, int nvar, int nfun, int *pdd);
```

The arguments of these functions are the following

- The first argument is an integer number representing the index of the variable or the index of the extra function. This index is zero based. i means the $(i+1)$-th variable or extra function.
- The second argument is an string of characters that represents the derivative. Let us suppose we differentiate with respect to four elements (intial conditions or parameters) named $\alpha, \beta, \gamma, \delta$, then the symbol `"1/2/0/1"` represents the derivative $\partial^4/\partial\alpha\partial\beta^2\partial\delta$. The string `"0/0/0/0"` means no derivative, and when it is used in `position_variable` or `position_function` gives the column position of the variable or the extra function. If we differentiate with respect to only one variable the separator `"/"` may be omitted.
- The third and fourth argument are the number of variables of the ODE and the number of *extra* functions. They are the same arguments used when we call the integrator.
- The last argument `pdd` is a pointer to an integer that represents an array with the necessary information to compute the desired partial derivatives.

13.4 LibTIDES functions to compute events

LibTIDES has eight different functions to compute events. Four in double precision

```
void dp_tides_find_zeros(DBLinkedFunction fcn,
         int nvar, int npar, double *x, double *p,
         double tini, double tend, double tol,
         int *numevents, dp_data_matrix *dmat, FILE* fileout) ;

void dp_tides_find_extrema(DBLinkedFunction fcn,
         int nvar, int npar, double *x, double *p,
         double tini, double tend, double tol,
         int *numevents, dp_data_matrix *dmat, FILE* fileout) ;

void dp_tides_find_minimum(DBLinkedFunction fcn,
         int nvar, int npar, double *x, double *p,
```

```
         double tini, double tend, double tol,
         int *numevents, dp_data_matrix *dmat, FILE* fileout) ;

void dp_tides_find_maximum(DBLinkedFunction fcn,
         int nvar, int npar, double *x, double *p,
         double tini, double tend, double tol,
         int *numevents, dp_data_matrix *dmat, FILE* fileout) ;
```

and four in multiple precision

```
void mp_tides_find_zeros(MPLinkedFunction fcn,
         int nvar, int npar, mpfr_t *x, mpfr_t *p,
         mpfr_t tini, mpfr_t tend, mpfr_t tol,
         int *numevents, mp_data_matrix *dmat, FILE* fileout) ;

void mp_tides_find_extrema(MPLinkedFunction fcn,
         int nvar, int npar, mpfr_t *x, mpfr_t *p,
         mpfr_t tini, mpfr_t tend, mpfr_t tol,
         int *numevents, mp_data_matrix *dmat, FILE* fileout) ;

void mp_tides_find_minimum(MPLinkedFunction fcn,
         int nvar, int npar, mpfr_t *x, mpfr_t *p,
         mpfr_t tini, mpfr_t tend, mpfr_t tol,
         int *numevents, mp_data_matrix *dmat, FILE* fileout) ;

void mp_tides_find_maximum(MPLinkedFunction fcn,
         int nvar, int npar, mpfr_t *x, mpfr_t *p,
         mpfr_t tini, mpfr_t tend, mpfr_t tol,
         int *numevents, mp_data_matrix *dmat, FILE* fileout) ;
```

The arguments in all cases represent the same elements:

- *The linked function:* `fcn` is a pointer to the function that contains the iterations needful to integrate the ODE and obtain events. In this argument we write the name used in the second argument of `TSMCodeFiles`.
- *The dimensions of the problem:* `nvar, npar` are two integer numbers that represent, respectively, the number of variables and the number of parameters.

- *Initial value of the variables:* `x` is a pointer to a `double` (`mpfr_t`) that represents an array with `nvar` elements.
- *Value of the parameters:* `p` is a pointer to a `double` (`mpfr_t`), or an array with `npar` elements. It has the value of the parameters.
- *Integration points:* the variables `tini`, `tend` represent the limits of the integration interval where TIDES search the events. `tini` is the point where we give the initial conditions. `tini` can be less or greater than `tend`.
- *Tolerance:* `tol` represents the tolerance in the numerical method to search zeros of polynomials.
- *Number of events:* A pointer to the integer `numevents`, that represents the maximum number of events that we search inside the integration interval. If we find all `numevents` events the integration ends before the final point of the interval. If we pass a value `numevents = 0`, TIDES search all the events inside the interval. In output the value of `numevents` is the number of found events.
- *Output of the integrator:* `dmat` is a pointer to a `dp_data_matrix` (or `mp_data_matrix`) type that represent a data matrix where the output events will be stored. `fileout` is a pointer to a `FILE` where the output will be written on.

13.5 Using data matrices to store the results

To store the results of the integration in a data matrix we have two new LibTIDES data type named `dp_data_matrix` and `mp_data_matrix`, together with the functions to handle it, and in MathTIDES the `TSMCodeFiles` option `DataMatrix`.

The new data types are declared in LibTIDES by means of the C structures

```
typedef struct dp_DM {
        int rows;
        int columns;
        double **data;
} dp_data_matrix;

typedef struct mp_DM {
        int rows;
        int columns;
        mpfr_t **data;
} mp_data_matrix;
```

The dimensions of the matrix are declared inside the LibTIDES taylor integrator. The number of rows corresponds to the number of points where the solution is computed (including the initial point as the first row). The number of columns must be sufficient to store, in this order

- The point t_i.
- The value of the variables in t_i: $\boldsymbol{x}(t_i)$.
- The value of the functions $G_i(t_i, \boldsymbol{x}(t_i), \boldsymbol{p})$ if `AddFunction` is used.
- The value of the partials derivatives if they are computed (see chapter 10).

Inside the LibTIDES integration the memory space to store the bidimensional array is created dynamically. LibTIDES do not free automatically the space of the data matrices. LibTIDES do not free automatically the space of the data matrices. After using a data matrix it is convenient to force LibTIDES to delete it. There are two functions to do that

```
void delete_dp_data_matrix(dp_data_matrix *dm);
void delete_mp_data_matrix(mp_data_matrix *dm);
```