Part II

Learning to use **TIDES** by mean of examples

Chapter 5

Integrating ODEs with TIDES

5.1 Seven steps to integrate ODEs with TIDES

To integrate ODEs with TIDES we need MATHEMATICA and a C (or FORTRAN) compiler. We may decompose the work to integrate the ODE in seven steps: four steps in MATHEMATICA and three steps with a C (or FORTRAN) compiler.

1. With MATHEMATICA:

   (M.1) Load the package MathTIDES (see 5.2).

   (M.2) Declare the work directory (see 5.3).

   (M.3) Declare the differential equation (see 5.4).

   (M.4) Write the C (or FORTRAN) code files to integrate the previously declared differential equation. In this step we may choose between four different TSM Integrators. There are two different files created with MathTIDES: the ODE file and the driver (main program). The ODE file must be written with MathTIDES because it uses internal algorithms very difficult to create manually. The driver may be written manually or by using MathTIDES (see 5.5).

2. With a C (or FORTRAN) compiler:

   (C.1) Compile the files generated with MathTIDES.

   (C.2) Link them with LibTIDES.

   (C.3) Run the executable.

5.2 Step (M.1): loading MathTIDES

The first step in MATHEMATICA is to load the package MathTIDES by writing

In[1]:=

```
<< MathTIDES`
```

5.3 Step (M.2): declaring the work directory

The files written by MathTIDES are saved on the default directory of MATHEMATICA (you can know this directory with the expression `Directory[]`).

The user may change the default directory by using the expression `SetDirectory`, for instance, changing the default directory to the directory where the local MATHEMATICA notebook is. Let us suppose we open a notebook that is inside the folder `Example`, and we write

In[2]:=

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

Out[2]=

```
....../Example/
```

then, all the files created after this command are stored on this directory. The output of the previous command gives us the complete path of the new work directory.

5.4 Step (M.3): declaring the differential equation

The Taylor Series Method integrates only first order ODE systems. However, a higher order ODE, under 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 (see 6.2).
- `NthOrderODE` : declares a first order ODE from a n-th order ODE (see 8.1.1).
- `PotentialToODE` : declares a first order ODE from a potential function V (see 7.1.1).
- `HamiltonianToODE` : declares a first order ODE from a hamiltonian function $\mathcal{H}$ (see 8.2.1).

We will learn the use of the four expressions in the following chapters of examples. The result in all cases is an expression with head `FirstOrderODE$`, that contains the MathTIDES internal representation of a first order differential equation. It has the following four 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.
- *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.
- *Fourth argument:* the list $\{p_1, \ldots, p_m\}$ of symbols that represents the parameters. This is an empty list when the ODE has no parameter.

5.5 Step (M.4): writing the code files

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 with one of the previously described expressions.
- *Second argument:* an string that represents the 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.

Let's suppose that we write `"name"` as the second argument of `TSMCodeFiles`, then two different kind of files can be created: a driver (main problem), named `"dr_name.c"`, that contains a call to the integrator, and a file `"name.c"` (with its corresponding header file `"name.h"`) that contains the code of the ODE. When we use the minimal version in FORTRAN (minf-tides) the files have the extension `".f"` and no header file is written.

Compiling the driver and the ODE codes and linking them with LibTIDES (and with GMP and MPFR, when we use the version mp-tides) we obtain the executable to integrate the ODE (steps (C.1), (C.2)).

The expression `TSMCodeFiles` shows on the MATHEMATICA session the names of the written files and the directories where they have been stored.

The options of the third argument follow the general rules of MATHEMATICA:

- The format is: `NameOfTheOption -> ValueOfTheOption`
- There are several option for the expression `TSMCodeFiles`, and the order of this options it is indifferent.
- If we do not write a particular option it takes the default value.

There are two kind of options of `TSMCodeFiles`: the options that changes the differential equation and the options that changes the driver. In the next chapters we will describe all these options, together with the examples.

5.6 Steps (C.1), (C.2), (C.3): compiling, linking and running the code files

These steps will we explained in the next chapter 6

Chapter 6

Using the four versions of the **TSM Integrator**: the sine and cosine differential equation

6.1 Example: the sine and cosine differential equation

We begin with the simple differential equation

$$\frac{dx}{dt} = y, \quad \frac{dy}{dt} = -x, \tag{6.1}$$

with two variables and no parameters. The analytical solution of this differential equation with the initial conditions $x(0) = 0$, $y(0) = 1$ are the functions $x = \sin t$, $y = \cos t$.

In this chapter we will learn the basic use of TIDES by integrating this differential equation. Our first objective is to show on the screen the values of the sine and cosine in the points $\{0, \pi/4, \pi/2\}$.

This is a simple example of first order ODE of two variables and no parameter. The first thing to do is to learn how to declare a general first order ODE and apply it to the example.

6.2 Declaring first order differential equations

A first order ODE is represented by the equation

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}(t, \boldsymbol{y}(t); \boldsymbol{p}), \quad \boldsymbol{y}(t_0) = \boldsymbol{y}_0, \quad \boldsymbol{y} \in \mathbb{R}^n, \quad \boldsymbol{p} \in \mathbb{R}^m, \tag{6.2}$$

where

- t is the independent variable. It may appear explicitly, or not, in the function $\boldsymbol{f}$.
- $\boldsymbol{y} = (y_1, \ldots, y_n)$ is the n-dimensional vector of variables ($n > 0$).

- $\boldsymbol{p} = (p_1, \dots, p_m)$ is the m-dimensional vector of parameters ($m \geq 0$).
- $\boldsymbol{f} = (f_1, \dots, f_n)$ 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 $\{f_1, \dots, f_n\}$ 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 $\{y_1, \dots, y_n\}$ 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 $\{p_1, \dots, p_m\}$ 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.

In our example, to declare the sine and cosine differential equation we write

In[3]:=

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

Henceforth, we name `sincosODE` to this ODE. The arguments to declare the ODE are: the expressions of the right hand term of the differential equation `{y, -x}`, the symbol of the independent variable `t` (a dummy variable in this case), and the list of symbols of the variables `{x, y}`. In this case there is not fourth argument because there is no parameter.

6.3 Declaring the work directory

First of all we change in MathTIDES the work directory: The file `sincosODE.nb` with all the examples of this chapter is stored inside the directory `chapter06` that is in `TIDESExamples`.

```
In[4]:=

SetDirectory[NotebookDirectory[]]

Out[4]=

/....../TIDESExamples/chapter06
```

where the dots depend on where we copy the directory `TIDESExamples`.

We change too, in the shell terminal, the directory of work before to compile and run the code files.

```
$cd /....../TIDESExamples/chapter06
```

6.4 Options of `TSMCodeFiles` to declare the **TSM Integrator**

The last step in MathTIDES is to write the code to integrate the ODE. To do that we will use the MathTIDES expression `TSMCodeFiles`. In 5.5 we explained the two first arguments. Here we will see any of the options used to write the driver. Now we will learn the first options of this expressions: the options to choose the TSM Integrator.

6.4.0.1 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.

6.4.0.2 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.

6.4.0.3 *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.

6.5 More options of TSMCodeFiles to change the driver

6.5.0.4 *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.

In our problem we write `InitialConditions -> {0, 1}` , because the initial conditions are $x(0) = 0, y(0) = 1$.

6.5.0.5 *Option*: `Output`

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

```
     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` is written on the screen. In our case we want to write the solution on the screen, then we write `Output -> Screen`.

Finally we will write the option `IntegrationPoints -> {0, Pi/2, Points[1]}` to declare the integration points. This option will we explained in detail in *7.4.0.7*.

6.6 Integrating `sincosODE` with minf-tides

After declaring the work directory (see 6.3), we write in MathTIDES

```
In[5]:=
TSMCodeFiles[sincosODE,
  "sincosMFL",
  InitialConditions -> {0, 1},
  IntegrationPoints -> {0, Pi/2, Points[1]},
  Output -> Screen,
  MinTIDES -> "Fortran"]
```

Out[5]=

```
File "dr_sincosMFL.f", "sincosMFL.f" written on directory
"/....../TIDESExamples/chapter06".
```

The option `MinTIDES -> "Fortran"` writes a code that use the minf-tides integrator. Two Fortran files (with extension `.f`) are created.

Finally to integrate the ODE we open the terminal and, after changing the work directory, we compile the files with the Fortran compiler, we link them with LibTIDES and we run the executable.

```
$gfortran dr_sincosMFL.f  sincosMFL.f -lTIDES -lm -o sincosmfl
$./sincosmfl
```

Finally, the solution appears on the screen

```
0.0000000000000000E+00   0.0000000000000000E+00   0.1000000000000000E+01
0.1570796326794897E+01   0.1000000000000000E+01  -0.2220446049250313E-15
```

Each line of the output represents: $t_i, x(t_i), y(t_i)$.

Another way to do that, without linking the library LibTIDES, is by adding the option `TIDESFiles->True`.

In[6]:=

```
TSMCodeFiles[sincosODE,
  "sincosMF",
  InitialConditions -> {0, 1},
  IntegrationPoints -> {0, Pi/2, Points[1]},
  Output -> Screen,
  MinTIDES -> "Fortran",
  TIDESFiles->True]
```

Out[6]=

```
Files "dr_sincosMF.f", "sincosMF.f", "minf_tides.f" written
on directory "/....../TIDESExamples/chapter06".
```

then a new file named `minf_tides.f` is created. This file contains the integrator core and substitutes the library LibTIDES.

```
$gfortran dr_sincosMF.f  sincosMF.f minf_tides.f -lm -o sincosmf
$./sincosmf
```

6.7 Integrating `sincosODE` with minc-tides

After declaring the work directory (see 6.3), we write in MathTIDES

In[7]:=

```
TSMCodeFiles[sincosODE,
  "sincosMCL",
  InitialConditions -> {0, 1},
  IntegrationPoints -> {0, Pi/2, Points[1]},
  Output -> Screen,
  MinTIDES -> "C"]
```

Out[7]=

```
Files "dr_sincosMCL.c", "sincosMCL.c", "sincosMCL.h" written
on directory "/....../TIDESExamples/chapter06".
```

The option `MinTIDES -> "C"` writes a code that use the minc-tides integrator. Three C files (with extensions `.c` and `.h`) are created.

Finally to integrate the ODE we open the terminal and after changing the work directory we compile the files with the C compiler, link them with LibTIDES and run the executable and the screen shows the solution.

```
$gcc dr_sincosMCL.c  sincosMCL.c -lTIDES -lm -o sincosmcl
$./sincosmcl
```

Another way to do that, without linking the library LibTIDES, is by adding the option `TIDESFiles->True`.

In[8]:=

```
TSMCodeFiles[sincosODE,
  "sincosMC",
  InitialConditions -> {0, 1},
  IntegrationPoints -> {0, Pi/2, Points[1]},
  Output -> Screen,
  MinTIDES -> "C",
  TIDESFiles->True]
```

Out[8]=

```
Files "dr_sincosMC.c", "sincosMC.c", "sincosMC.h", "minc_tides.c",
"minc_tides.h" written on directory "/....../TIDESExamples/chapter06".
```

then a new file named `minc_tides.c` is created. This file contains the integrator core and substitutes the library LibTIDES.

```
$gcc dr_sincosMC.c  sincosMC.c minc_tides.c -lm -o sincosmc
$./sincosmc
```

6.8 Integrating `sincosODE` with **dp-tides**

After declaring the work directory (see 6.3), we write in MathTIDES

In[9]:=

```
TSMCodeFiles[sincosODE,
  "sincosDP",
  InitialConditions -> {0, 1},
  IntegrationPoints -> {0, Pi/2, Points[1]},
  Output -> Screen]
```

Out[9]=

```
Files "dr_sincosDP.c", "sincosDP.c", "sincosDP.h" written on directory
"/....../TIDESExamples/chapter06".
```

If we do not use the option `MinTIDES` the standard option is created, in this case we call the dp-tides integrator. Three C files (with extensions `.c` and `.h`) are created.

Finally to integrate the ODE we open the terminal and after changing the work directory we compile the files with the C compiler, link them with LibTIDES, and run the executable.

```
$gcc dr_sincosDP.c  sincosDP.c -lTIDES -lm -o sincosdp
$./sincosdp
```

6.9 Integrating `sincosODE` with mp-tides

After declaring the work directory (see 6.3), we write in MathTIDES

In[10]:=

```
TSMCodeFiles[sincosODE,
  "sincosMP",
  InitialConditions -> {0, 1},
  IntegrationPoints -> {0, Pi/2, Points[1]},
  Output -> Screen,
  Precision -> Multiple[30]]
```

Out[10]=

```
Files "dr_sincosMP.c", "sincosMP.c", "sincosMP.h" written on directory
"/....../TIDESExamples/chapter06".
```

With the option `Precision -> Multiple[30]` we call to the mp-tides integrator and prepare the integrator to work with 30 digits of precision. Three C files (with extensions `.c` and `.h`) are created.

Finally to integrate the ODE we open the terminal and after changing the work directory we compile the files with the C compiler, link them with LibTIDES and the MPFR and GMP libraries, and run the executable.

```
$gcc dr_sincosMP.c  sincosMP.c -lTIDES -lm -lmpfr -lgmp -o sincosmp
$./sincosmp
```

Let's note that the option `Precision` is not compatible with the option `MinTIDES`.

Chapter 7

Understanding the files written by **MathTIDES**: the keplerian motion

7.1 The keplerian motion

The keplerian motion can be described by the second order ODE

$$\ddot{x} = -\frac{\partial V}{\partial x}, \quad \ddot{y} = -\frac{\partial V}{\partial y}, \quad \ddot{z} = -\frac{\partial V}{\partial z}, \tag{7.1}$$

where the potential function V is

$$V(x, y, z) = -\frac{\mu}{\sqrt{x^2 + y^2 + z^2}}, \tag{7.2}$$

and μ represents a parameter.

By defining three new variables $X = \dot{x}, Y = \dot{y}, Z = \dot{z}$ we may write the equations (7.1) as a first order ODE

$$\dot{X} = -\frac{\partial V}{\partial x}, \quad \dot{Y} = -\frac{\partial V}{\partial y}, \quad \dot{Z} = -\frac{\partial V}{\partial z}, \quad \dot{x} = X, \quad \dot{y} = Y, \quad \dot{z} = Z \tag{7.3}$$

With MathTIDES it is sufficient to declare the expression of the potential function and it extends the variables and computes the derivatives to obtain explicitely the differential equation (7.3).

7.1.1 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 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 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.

`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. The symbols of the new variables (derivatives) are formed by adding `$d1` to the symbol of the duplicated variables.

In our case to obtain (7.3) from (7.2) we write

In[11]:=

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

Out[11]=

```
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}]
```

The symbols that represent the variables of the first order ODE are in this case: {`x, y, z, x$d1, y$d1, z$d1`}.

Let us suppose the position and velocity of the orbiter in the initial instant are $\boldsymbol{x} = (0.8, 0, 0), \boldsymbol{X} = (0, 1.2247448713915892, 0)$, and we choose a set of units such as $\mu = 1$. In this conditions the period of the orbit is equal to 2π. Then we want to compute the position and velocity in five points between $t = 0$ and the period $T = 2\pi$ in equidistant time intervals. Then, in MathTIDES we write

In[12]:=

```
TSMCodeFiles[keplerODE,
 "kepler",
 InitialConditions -> {0.8, 0, 0, 0, 1.2247448713915892, 0},
 ParametersValue -> {1},
 IntegrationPoints -> {0, 2 Pi, Points[4]},
 Output -> Screen]

Out[12]=

"Files "dr_kepler.c", "kepler.h", kepler.c", written on directory
"/....../TIDESExamples/chapter07".
```

In this example we have a parameter. We give the value of the parameter with the option

7.1.1.6 *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.

7.2 Understanding the driver

The driver is the main program, where we declare the parameters of the integrator and we call it. MathTIDES writes the most simple driver with the essential information, but, understanding this driver, the user can write or change it manually. In what follows we enumerate the essential points of the driver of the previous example:

1. It includes the TIDES header file (`dp_tides.h` for double precision and `mp_tides.h` for multiple precision) and the ODE header file. After that the main function begins.

```
#include "dp_tides.h"
#include "kepler.h"


int main() {
```

2. It declares the parameters of the ODE

```
        int npar = 1;
        double p[npar];
        p[0] = 1. ;
```

To initialize the parameters it declares a `int` variable `npar` with the number of parameters (in our case `npar=1`), and a double array, `double p[npar]` whose dimension coincides with the number of parameters. Finally it gives value to the parameters.

3. It declares the variables of the ODE

```
        int nvar = 6;
        double v[nvar];
        v[0] = 0.8 ;
        v[1] = 0.0 ;
        v[2] = 0.0 ;
        v[3] = 0.0 ;
        v[4] = 1.2247448713915892 ;
        v[5] = 0.0 ;
```

To initialize the variables it declares a `int` variable `nvar` with the number of variables, six in our case: three variables (position) and three first order derivatives (velocity), and a double array, `double v[nvar]` whose dimension coincides with the number of variables. Finally it gives value to the variables.

4. It declares the number of extra functions

```
        int nfun = 0;
```

Besides the evolution with the time of the variables, with TIDES we may compute the evolution of functions of the variables. We will explain in detail this option in (8.2.3). If we do not use this possibility we declare a `int` variable `nfun` equal to 0.

5. It declares the tolerances (relative and absolute) used in the numerical integration

```
        double tolrel = 1.e-16 ;
        double tolabs = 1.e-16 ;
```

By default the driver defines both tolerances equal to 10^{-16}. The user may change them manually on the driver or by using the options: `RelativeTolerance -> rtvalue`, `AbsoluteTolerance -> atvalue` in MathTIDES.

6. It declares the integration points

```
        double tini = 0.0;
        double tend = 6.283185307179586;
        int    nipt = 4;
        double dt = (tend - tini)/nipt ;
```

The way to declare the integration points is discussed in detail in section (7.4) in this chapter.

7. It declares the output way

```
        FILE* fd = stdout;
```

In this case the driver uses the standard output (screen) by declaring the pointer to `FILE fd`.

8. It calls the LibTIDES funtion `dp_tides_delta` to integrate the problem

```
        dp_tides_delta(kepler, NULL, nvar, npar, nfun, v, p,
                       tini, dt, nipt, tolrel, tolabs, NULL, fd);
```

This, and other LibTIDES functions to integrate ODES, and their arguments, will we described in detail in (7.5).

9. It ends the main program

```
        return 0;
}
```

7.3 Understanding the ODE file

The user may change easily the driver, however the ODE file needs to be created by MathTIDES. Here we find a description of the ODE file for the integration of the Kepler problem

1. It has a block of initialization of the variables. This block is always identical except for the name and the value of the variables.

```
long  kepler(iteration_data *itd, double t, double v[],
                             double p[],  int ORDER, double *cvfd)
{
        int i;
        static int   VARIABLES        = 6;
        static int   PARAMETERS       = 1;
        static int   FUNCTIONS        = 0;
        static int   LINKS            = 13;
        static int   POS_FUNCTIONS[1] = {0};
        initialize_dp_case();
        double ct[] = {-1.5, -1.};
```

2. It has a block with the ODE function after applying the automatic differentiation rules

```
        for(i=0;  i<=ORDER; i++) {
                double_var_t(itd, var[4],var[1], i);
                double_var_t(itd, var[5],var[2], i);
                double_var_t(itd, var[6],var[3], i);
                double_var_t(itd, link[10],var[4], i);
                double_var_t(itd, link[11],var[5], i);
                double_var_t(itd, link[12],var[6], i);
                double_mul_t_cc(itd, ct[1],par[0],link[0],i);
                double_mul_t(itd, var[1],var[1],link[1],i);
                double_mul_t(itd, var[2],var[2],link[2],i);
                double_mul_t(itd, var[3],var[3],link[3],i);
                double_add_t(itd, link[1],link[2],link[4],i);
                double_mul_t(itd, link[0],var[1],link[5],i);
                double_mul_t(itd, link[0],var[2],link[6],i);
                double_mul_t(itd, link[0],var[3],link[7],i);
                double_add_t(itd, link[3],link[4],link[8],i);
                double_pow_t_cc(itd, link[8],ct[0],link[9],i);
                double_mul_t(itd, link[5],link[9],link[10],i);
                double_mul_t(itd, link[6],link[9],link[11],i);
                double_mul_t(itd, link[7],link[9],link[12],i);
        }
```

3. It has an ending block. This is always identical

```
        write_dp_solution();
        return NUM_COLUMNS;
}
```

7.4 Ways to declare the integration points

The points where the result of the integration are showed or stored may be handled by means of an option of MathTIDES.

7.4.0.7 *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.
  - 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 (increasing or decreasing). They can be non-equidistant points.

In our example we have four ways to declare the integration points

1. `IntegrationPoints -> {0, Delta[Pi/2], Points[4]}`

```
TSMCodeFiles[keplerODE,
     "kepler1",
     InitialConditions -> {0.8, 0, 0, 0, 1.2247448713915892, 0},
     ParametersValue -> {1},
     IntegrationPoints -> {0, Delta[Pi/2], Points[4]},
     Output -> Screen]
```

   With the option `IntegrationPoints -> {0, Delta[Pi/2], Points[4]}` we integrate in five (4+1) points: the first point is the first element (0), and a value of $\pi/2$ between each point. By using this option the part of the driver that declares the integration points has the format

```
        double tini = 0.0;
        double dt   = 1.570796326794897;
        int    nipt = 4;
```

   In the driver we declare the initial point in a `double` variable `tini`, The increment in a `double` variable `dt`, and the number of points (without counting the initial point t_0) in a `int` variable `nipt`.

2. `IntegrationPoints ->{0, 2 Pi , Points[4]}`

```
TSMCodeFiles[keplerODE,
     "kepler2",
     InitialConditions -> {0.8, 0, 0, 0, 1.2247448713915892, 0},
     ParametersValue -> {1},
     IntegrationPoints -> {0, 2 Pi , Points[4]},
     Output -> Screen]
```

With the option `IntegrationPoints -> {0, 2 Pi, Points[4]}` we integrate in five (4+1) equidistant points between 0 and 2π. In the driver we change the block that declares the integration points by adding the final point in a `double` variable `tend` and computing `dt` instead of declaring it.

```
        double tini = 0.0;
        double tend = 6.283185307179586;
        int    nipt = 4;
        double dt = (tend - tini)/nipt ;
```

3. `IntegrationPoints -> {0, 2 Pi, Delta[Pi/2]}`

```
TSMCodeFiles[keplerODE,
     "kepler3",
     InitialConditions -> {0.8, 0, 0, 0, 1.2247448713915892, 0},
     ParametersValue -> {1},
     IntegrationPoints -> {0, 2 Pi, Delta[Pi/2]},
     Output -> Screen]
```

With the option `IntegrationPoints -> {0, 2 Pi, Delta[Pi/2]}` we integrate in five (4+1) equidistant points between in increments of $\pi/2$. In the driver we change the block that declares the integration points by declaring the initial, final points and the increment and computing the number of points.

```
        double tini = 0.0;
        double dt   = 1.570796326794897;
        double tend = 6.283185307179586;
        int    nipt = (int) floor ((tend-tini)/dt);
```

4. `IntegrationPoints -> {0, Pi/2, Pi , 3 Pi/2, 2 Pi}`

```
TSMCodeFiles[keplerODE,
     "kepler4",
     InitialConditions -> {0.8, 0, 0, 0, 1.2247448713915892, 0},
     ParametersValue -> {1},
     IntegrationPoints -> {0, Pi/2,  Pi , 3 Pi/2, 2 Pi},
```

With the option `IntegrationPoints -> {0, Pi/2, Pi , 3 Pi/2, 2 Pi}` we declare a list of points where the solution is computed. The driver in this case is different: we need to declare the total number of points in the `int` variable `ntot`, declare the array `double lt[ntot]` with dimension equal to the number of points and eventually declare the elements of the array.

```
        int ntot  = 5;
        double lt[ntot] ;
        lt[0] = 0.0 ;
        lt[1] = 1.570796326794897 ;
        lt[2] = 3.141592653589793 ;
        lt[3] = 4.71238898038469 ;
        lt[4] = 6.283185307179586 ;
```

In this case the driver changes, not only the way to declare the integration points, but the way to call the integrator. We use a new LibTIDES function `dp_tides_list` whose arguments will be described in the next section.

```
        dp_tides_list(kepler, NULL, nvar, npar, nfun,
                       v, p, lt, ntot, tolrel, tolabs, NULL, fd);
```

7.5 LibTIDES functions to call the integrator

There are two LibTIDES functions to call the TSM Integrator.

```
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) ;
```

The arguments of both 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` 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. If there is no parameter this argument will be `NULL`.
- *Integration points(case* `dp_tides_delta` *):* the integration points are represented by three arguments: two `double` variables `tini, dt` that contains 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(case* `dp_tides_list` *):* the integration points are represented by two arguments `lt` and `ntot`. `lt` is a pointer to a `double` 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` variables with the relative and absolute tolerance of the method.

- *Output of the integrator:* `dmat` is a pointer to a `dp_data_matrix` type that represent a data matrix where the output will be stored (this will be explained later on section 8.2.4). `fileout` is a pointer to a `FILE` where the output will be written on.

Chapter 8

More examples: several gravitational problems

8.1 The three body problem

The planar three body problem is formulated by means of the second order differential equations

$$\ddot{x} - 2\dot{y} = x - \frac{(1-\mu)(x+\mu)}{(\sqrt{(x+\mu)^2+y^2})^3} - \frac{\mu(x+\mu-1)}{(\sqrt{(x+\mu-1)^2+y^2})^3},$$

$$\ddot{y} + 2\dot{x} = y - \frac{(1-\mu)y}{(\sqrt{(x+\mu)^2+y^2})^3} - \frac{\mu y}{(\sqrt{(x+\mu-1)^2+y^2})^3},$$

where (x, y) and $(\dot{x}, \dot{y})$ represent the position and velocity of the third body with respect to the keplerian orbit of the primaries, and $\mu \in (0, 1)$ represents the ratio of masses of the primaries.

MathTIDES handles k-th order differential equations by transforming them into first order ODEs.

8.1.1 Higher order differential equations

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

$$\boldsymbol{F}(t, \boldsymbol{y}, \frac{d\boldsymbol{y}}{dt}, \frac{d^2\boldsymbol{y}}{dt^2}, \dots, \frac{d^k\boldsymbol{y}}{dt^k}; \boldsymbol{p}) = 0, \qquad \boldsymbol{y}(t_0) = \boldsymbol{y}_0, \dots, \frac{d^k\boldsymbol{y}}{dt^k}(0) = \boldsymbol{y}_0^{(k)}, \tag{8.1}$$

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

Let us suppose that all the derivatives $y_1^{(k)}, \dots y_n^{(k)}$ of the greatest order k appear explicitely in (8.1), then, solving the system (8.1) in $y_1^{(k)}, \dots y_n^{(k)}$, if it is possible, we transform the k-th order ODE into a first order ODE by introducing the derivatives $\frac{d\boldsymbol{y}}{dt}, \frac{d^2\boldsymbol{y}}{dt^2}, \dots, \frac{d^{k-1}\boldsymbol{y}}{dt^{k-1}}$ 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 have 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 (maintaining the relative order of the variables)
3. Second derivatives (maintaining the relative order of the variables)

4. ....

To illustrate the use of `NthOrderODE` we will see in the next subsection the three body example. There are two more examples in (12.1.2).

8.1.2 Finding a horseshoe orbit

A horseshoe orbit is a particular solution of the three body problem. The tutorial will be continued by finding and plotting one of these horseshoe orbits. First of all we define the ODE

In[13]:=

```
threeBP = {
   x'' - 2 y' == x - (1 - mu) (x + mu)/r^3 - mu (x + mu - 1)/s^3,
   y'' + 2 x' == y - (1 - mu) y/r^3 - mu y/s^3
} /. {r -> Sqrt[(x + mu)^2 + y^2], s -> Sqrt[(x + mu - 1)^2 + y^2]};
```

In[14]:=

```
threeBPEQ = NthOrderODE[threeBP, t, {x, y}, {mu}]
```

Out[14]=

```
FirstOrderODE$[{x$d1, y$d1,
   x - (mu (-1 + mu + x))/((-1 + mu + x)^2 + y^2)^(
       3/2) - ((1 - mu) (mu + x))/((mu + x)^2 + y^2)^(3/2) + 2 y$d1,
   -2 x$d1 + y - (mu y)/((-1 + mu + x)^2 + y^2)^(
       3/2) - ((1 - mu) y)/((mu + x)^2 + y^2)^(3/2)},
   t, {x, y, x$d1, y$d1}, {mu}]
```

where we see that the variables are, in this order, the position and the first derivatives, `{x, y, x$d1, y$d1}`.

To find the horseshoe orbit we integrate this ODE with the initial conditions ($x = 0.85, y = 0.5, \dot{x} = 0, \dot{y} = 0$), and a value of the parameter $\mu = 0.001$. Instead of writing the solution on the screen we write it into a file named `horseshoe`. To create the files, with the driver, we write in MathTIDES

In[15]:=

```
TSMCodeFiles[threeBPEQ,
      "threebody",
      InitialConditions -> {0.85, 0.5, 0, 0},
      ParametersValue -> {0.001},
      IntegrationPoints -> {0, 150, Points[150]},
      Output -> "horseshoe"]
```

Out[15]=

```
Files "dr_threebody.c", "threebody.h", threebody.c", written
on directory "/....../TIDESExamples/chapter08".
```

The main difference with respect to previous examples is the use of the `Output` option. In this case a string `"horseshoe"` means that the solution will be written into a file with the name of the string. The file has the same format that the screen output:

- Each line represents an instant (point) where the solution is computed.
- If there are n variables, the output has $n+1$ columns. The first column is the time t_i where the solution is computed and the rest of columns represent the value of the variables in t_i.

With MATHEMATICA, or any plotting software, we may read the file and plot the solution

In[16]:=

```
dat = OpenRead["horseshoe"];
sol = ReadList[dat, {Real, Real, Real, Real, Real}];
solxy = Map[{#[[2]], #[[3]]} &, sol];
ListPlot[solxy, Joined -> True, AspectRatio -> Automatic]
```

and we obtain the plot

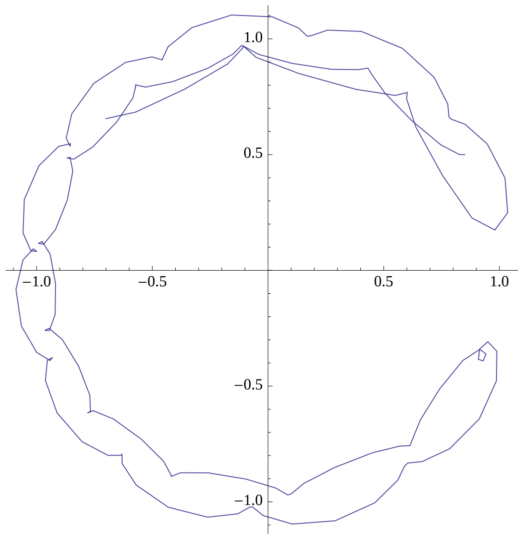

Figure 8.1: Horseshoe orbit

8.2 The main problem of the Earth artificial satellite

The first approximation to the motion of a Earth artificial satellite is the keplerian motion given by the Hamiltonian $\mathcal{H}_k = v^2/2 - \mu/r$, where the position (x, y, z) represents the coordinates, the velocity (X, Y, Z) represents the momenta, $r = \sqrt{x^2 + y^2 + z^2}, v = \sqrt{X^2 + Y^2 + Z^2}$, and μ is the the gravitational constant.

The non-sphericity of the Earth perturbs this motion by adding a term to the hamiltonian that is represented by an infinite series that depends on a set of constants named the harmonics. The most important harmonic, J_2, is the term due to the flattening of the Earth. If we consider only this term, by taking the rest of harmonics equal to zero, we have a better approximation to the motion of a satellite named *the main problem*.

The Hamiltonian of the main problem is then

$$\mathcal{H} = \mathcal{H}_k + V_z, \quad \mathcal{H}_k = \frac{v^2}{2} - \frac{\mu}{r}, \quad V_z = \frac{\mu J_2 a^2}{r^3} P_2(\frac{z}{r}). \tag{8.2}$$

where a is the equatorial radius, and $P_2()$ represent the Legendre polynomial of order two.

The differential equations of this problem are given by the Hamilton's equations

$$\begin{aligned} \dot{x} &= \frac{\partial \mathcal{H}}{\partial X}, & \dot{y} &= \frac{\partial \mathcal{H}}{\partial Y}, & \dot{z} &= \frac{\partial \mathcal{H}}{\partial Z}, \\ \dot{X} &= -\frac{\partial \mathcal{H}}{\partial x}, & \dot{Y} &= -\frac{\partial \mathcal{H}}{\partial y}, & \dot{Y} &= -\frac{\partial \mathcal{H}}{\partial x}. \end{aligned}$$

We need to differentiate the hamiltonian with respect to both, variables and momenta, to construct the equations. With MathTIDES it is sufficient to declare the expression of the Hamiltonian and it computes the derivatives to obtain the differential equation.

Let us suppose we use the equatorial radius of the Earth as the length unit, and the minute as the time unit, then the value of the parameters are $(0.005530428042714393, 1, 0.0010826266835531513)$. With this election we take a set of initial conditions in wich the variables are given by the vector $\boldsymbol{x} = (1.3, 0, 0)$, and the momenta are given by the vector $\boldsymbol{X} = (0, 0.06423314045257492, 0.011326035717425298)$, that in the Keplerian problem correspond with an orbit of period equal to 125.232059785382 minutes. We want the solution in five points each 25 minutes.

8.2.1 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

$$\frac{d\boldsymbol{x}}{dt} = \frac{\partial \mathcal{H}}{\partial \boldsymbol{X}}, \quad \frac{d\boldsymbol{X}}{dt} = -\frac{\partial \mathcal{H}}{\partial \boldsymbol{x}}. \tag{8.3}$$

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.

8.2.2 The main problem ODE

In our example we compute the Hamiltonian `HamZ2` as the sum of the energy of the Keplerian problem `T+V` and the potential of the main problem `zon2`, then the ODE named `ztODE2` is given by the expression

```
In[17]:=

T = (X^2 + Y^2 + Z^2)/2 ;
V = -mu/Sqrt[x^2 + y^2 + z^2];
zon2 = (mu rt^2)/r^2 J2 LegendreP[2, z/r] /. r -> Sqrt[x^2 + y^2 + z^2];
HamZ2 = T + V + zon2;

ztODE2 = HamiltonianToODE[HamZ2, t, {x, y, z, X, Y, Z}, {mu, rt, J2}]


Out[17]=

FirstOrderODE$[{X, Y, Z,
  (J2 mu rt^2 x)/(x^2 + y^2 + z^2)^2 - (mu x)/(x^2 + y^2 + z^2)^(3/2) +
        (2 J2 mu rt^2 x (-x^2 - y^2 + 2 z^2))/(x^2 + y^2 + z^2)^3,
  ( J2 mu rt^2 y)/(x^2 + y^2 + z^2)^2 - (mu y)/(x^2 + y^2 + z^2)^(3/2) +
        (2 J2 mu rt^2 y (-x^2 - y^2 + 2 z^2))/(x^2 + y^2 + z^2)^3,
  -((2 J2 mu rt^2 z)/(x^2 + y^2 + z^2)^2) -
        (mu z)/(x^2 + y^2 + z^2)^(3/2) +
        (2 J2 mu rt^2 z (-x^2 - y^2 + 2 z^2))/(x^2 + y^2 + z^2)^3},
  t, {x, y, z, X, Y, Z}, {mu, rt, J2}]
```

with three parameters {`mu, rt, J2`}.

8.2.3 Computing *extra* functions

With this example we are not only interested into the values of the solution $\boldsymbol{x}(t_i), \boldsymbol{X}(t_i)$ at the desired points. We want to know the evolution of the functions $T+V$ and $\mathcal{H}$ to check how the energy of the Keplerian problems evolves in this problem, and how the energy of the system is maintained during the integration. To do that we use, in `TSMCodeFiles`, the option `AddFunctions`.

8.2.3.8 Option: `AddFunctions`

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, ...

8.2.4 Using data matrices to store the result

Instead of declaring the screen or a file to write the solution we want, in this case, to store it into a data matrix to use it in later operations of the main program. To do that we have a new LibTIDES data type named `dp_data_matrix`, together with the functions to handle it, and in MathTIDES the `TSMCodeFiles` option `DataMatrix`.

The new data types `dp_data_matrix` is declared in LibTIDES by means of the C structure

```
typedef struct dp_DM {
        int rows;
        int columns;
        double **data;
} dp_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).

Let us suppose a data matrix named `dm`. Once `dm` has been initialized, we may obtain the number of rows and columns of this matrix by using `dm.rows`, `dm.columns`. The element (i, j) of the matrix is `dm.data[i][j]`.

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. After using a data matrix it is convenient to force LibTIDES to delete it by using the function

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

We may use a data matrix in the driver including the `TSMCodeFiles` option `DataMatrix`.

8.2.4.9 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 will be stored. The name is `nameDM` in the second case or the name of the file joined to `"_DataMatrix"` in the first case.

8.2.5 The integration code of the main problem

Then the C files to integrate the main problem of the satellite will be obtained by writing in MathTIDES

In[18]:=

```
TSMCodeFiles[ztODE2, "SatJ2",
    InitialConditions -> {1.3, 0, 0,
                     0, 0.06423314045257492, 0.011326035717425298},
    ParametersValue -> {0.005530428042714393, 1, 0.0010826266835531513},
    IntegrationPoints -> {0, 125, Delta[25]},
    DataMatrix -> "datj2",
    AddFunctions -> {(T + V), HamZ2}]
```

Out[18]=

```
Files "dr_SatJ2.c", "SatJ2.h", SatJ2.c", written on directory
"/....../TIDESExamples/chapter08"
```

If we read the file `dr_SatJ2.c` we observe several differences with respect to previous drivers. The first one is the line

```
        dp_data_matrix datj2;
```

that declares a `dp_data_matrix` to store the solution.

The address of this pointer is passed to `dp_tides_delta`, where the pointer is initialized with the adequate dimensions.

```
        dp_tides_delta(SatJ2, NULL, nvar, npar, nfun, v, p,
                       tini, dt, nipt, tolrel, tolabs, &datj2, NULL);
```

Let us note that the last argument of the driver is NULL because of we do not write the solution into any file nor on the screen.

The changes to compute the additional functions appear in the file SatJ2.c, i.e. the ODE file, not in the driver.

There are two ways to compute the energy of the system: internally by using the TIDES option to compute extra functions, and externally by computing the energy from the value of the variables. In this example we want to illustrate both ways, then we need to change manually the driver (dr_SatJ2M.c) by including the function

```
double energy(double *v, double *p)
{
        double r, cener, pener, j2ener, ener;
        r = sqrt(v[0]*v[0]+v[1]*v[1]+v[2]*v[2]);
        cener =  (v[3]*v[3]+v[4]*v[4]+v[5]*v[5])/2.;
        pener = -p[0]/r;
        j2ener = v[2]/r;
        j2ener = (3*j2ener*j2ener -1)/2.;
        j2ener = p[0]*p[1]*p[1]*p[2]*j2ener/(r*r);
        ener = cener+pener+j2ener;
        return ener;
}
```

that computes the energy from the solution.

By adding the next piece of code we show on the screen the values of T+V and H, computed inside the TIDES integrator, and the difference H-HC of the energy H computed inside TIDES and HC computed with the function energy.

```
        int i,j;
        double var[6], ener;
        for(i = 0 ; i <= nipt; i++) {
                for(j = 0; j <6; j++)  var[j] = datj2.data[i][j+1];
                ener = energy(var,p);
                printf("T+V = %.15le, H = %.15le, H - HC = %.10le\n",
                       datj2.data[i][7], datj2.data[i][8],
                       datosj2.data[i][8]-ener);
        }
```

To print these values and the difference H-HC we make use of the solution stored in datj2, where datj2.data[i][j] represents the row i (values of the solution in t_i), and the column j is the corresponding value. In this case we have

- Column 0: time t_i.
- Columns 1,2,3: the variables $x(t_i), y(t_i), z(t_i)$.
- Columns 4,5,6: the momenta $X(t_i), Y(t_i), Z(t_i)$.
- Column 7: The value of $T + V$ in t_i.
- Column 8: The value of $\mathcal{H}$ in t_i.

Finally we obtain the following results

```
T+V = -2.127087708736305e-03, H = -2.128859125591486e-03, H - HC =  0.0000000000e+00
T+V = -2.127230402142720e-03, H = -2.128859125591488e-03, H - HC = -8.6736173799e-19
T+V = -2.127137530248718e-03, H = -2.128859125591488e-03, H - HC = -4.3368086899e-19
T+V = -2.127138484455382e-03, H = -2.128859125591487e-03, H - HC = -8.6736173799e-19
T+V = -2.127229794041088e-03, H = -2.128859125591488e-03, H - HC =  4.3368086899e-19
T+V = -2.127087715068851e-03, H = -2.128859125591488e-03, H - HC =  4.3368086899e-19
```

Chapter 9

Handling multiple precision in TIDES: the elliptic integral of the first kind

9.1 LibTIDES and MPFR library

LibTIDES handles multiple–precision by using the MPFR library. It 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.

9.2 The elliptic integral of the first kind

Let us take the first order differential equation

$$\frac{dx}{dt} = \frac{1}{\sqrt{1-k^2\sin^2 t}}, \quad x(0) = 0. \tag{9.1}$$

whose solution, $x(t;k) \in \mathbb{R}\times\mathbb{R} \to \mathbb{R}$, is the elliptic integral of the first kind

$$x(t;k) = F(t;k) = \int_0^t \frac{1}{\sqrt{1-k^2\sin^2 s}}d\,s.$$

The ODE (9.1) is a non-autonomous system (the independent variable t appears on the ODE) and it has a parameter k. To declare this ODE in MathTIDES we write

In[19]:=

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

Let us suppose we want to compute the values of the first order differential equation, for $k = 0.5$ in the points $t = 0, \pi/8, \pi/4, 3\pi/8, \pi/2$, and we want to work with 30-digits of precision, then we create the driver and the ODE files by writing

In[20]:=

```
TSMCodeFiles[ellipFODE,
    "ellipF30",
    InitialConditions -> {0},
    ParametersValue -> {.5},
    IntegrationPoints -> {0, Delta[Pi/8], Points[4]},
    Precision->Multiple[30],
    Output -> Screen]
```

Out[20]=

```
"Files "dr_ellipF30.c", "ellipF30.h", ellipF30.c", written on directory
"/....../TIDESExamples/chapter09".
```

All the options used here have been explained previously.

9.3 Driver for multiple precision arithmetic

The driver is the main program where we declare the parameters of the integrator and we call it. MathTIDES writes the most simple driver with the essential information, but understanding this driver the user can write or change it manually. In what follows we enumerate the essential points of the driver of the previous example:

1. It includes the MPFR header file, the TIDES header file `mp_tides.h` and the ODE header file. After that the main function begins.

```
#include "mpfr.h"
#include "mp_tides.h"
#include "ellipF30.h"
```

```
int main() {
    int i;
```

2. It declares the decimal precision digits

```
    set_precision_digits(30);
```

With this function TIDES computes the value of TIDES_PREC used to declare each mpfr_t variable.

3. It declares the parameters

```
    int npar = 1;
    mpfr_t p[npar];
    for(i=0; i<npar; i++) mpfr_init2(p[i], TIDES_PREC);
    mpfr_set_str(p[0], "0.5", 10, TIDES_RND);
```

To initialize the parameters we declare a `int` variable `npar` with the number of parameters (one in our case), an array `mpfr_t p[npar]` whose dimension coincides with the number of parameters. We initialize each element of the array and finally we give value to the parameters. The value of the parameters is passed by means of an string that represent the value in base 10. MathTIDES writes this string with the desired precision. The program uses the default rounding mode `TIDES_RND`.

4. It declares the variables

```
    int nvar = 1;
    mpfr_t v[nvar];
    for(i=0; i<nvar; i++) mpfr_init2(v[i], TIDES_PREC);
    mpfr_set_str(v[0], "0", 10, TIDES_RND);
```

To initialize the variables we declare a `int` variable `nvar` with the number of variables (one in our case), an array, `mpfr_t v[nvar]` whose dimension coincides with the number of variables. We initialize each element of the array and finally we give value to the variables. The value of the variables is passed by means of an string that represent the value in base 10. MathTIDES writes this string with the desired precision.

5. It declares the number of extra functions

```
    int nfun = 0;
```

Besides the evolution with the time of the variables, with TIDES we may compute the evolution of functions of the variables. We will explain in detail this option in 8.2.3. If we do not use this possibility we declare a `int` variable `nfun` equal to 0.

6. It declares the tolerances (relative and absolute) used in the numerical integration

```
    mpfr_t tolrel, tolabs;
    mpfr_init2(tolrel, TIDES_PREC);
    mpfr_init2(tolabs, TIDES_PREC);
    mpfr_set_str(tolrel, "1.e-29", 10, TIDES_RND);
    mpfr_set_str(tolabs, "1.e-29", 10, TIDES_RND);
```

   By default the driver defines both tolerances equal to $10^{(1-p)}$where p is number of precision digits. The user may change them manually on the driver or by using the options: `RelativeTolerance -> rtvalue`, `AbsoluteTolerance -> atvalue` in MathTIDES.

7. It declares the integration points

```
    mpfr_t tini, dt;
    mpfr_init2(tini, TIDES_PREC);
    mpfr_init2(dt, TIDES_PREC);
    mpfr_set_str(tini, "0", 10, TIDES_RND);
    mpfr_set_str(dt,"0.392699081698724154807830422910",10,TIDES_RND);
    int  nipt  = 4;
```

   The way to declare the integration points is discussed in detail in the section 7.4. The code in multiple precision is similar to the code in `double` but we need to initialize the variables and give them a value by mean an string. In this case we see how the value of $\pi/8$ is computed in MathTIDES with 30 digits of precision and a string with this value is used to assign the value of `dt`.

8. It declares the output way

```
    FILE* fd = stdout;
```

   In this case the driver uses the standard output (screen) by declaring the pointer to `FILE fd`.

9. It calls to the LibTIDES function to integrate the problem

```
    mp_tides_delta(ellipF30, NULL, nvar, npar, nfun, v, p,
                        tini, dt, nipt, tolrel, tolabs, NULL, fd);
```

This, and other LibTIDES function to integrate ODEs, and their arguments, will we described in detail in 9.4.

10. It ends the main program

```
    return 0;
}
```

9.4 LibTIDES functions to call the integrator

There are two LibTIDES functions to call the TSM Integrator with multiple precision

```
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 both 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 `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 `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 (case* `mp_tides_delta` *):* the integration points are represented by three arguments: two `mpfr_t` variables `tini, dt` that contains 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 (case* `mp_tides_list` *):* the integration points are represented by two arguments `lt` and `ntot`. `lt` is a pointer to a `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 `mpfr_t` variables with the relative and absolute tolerance of the method.

- *Output of the integrator:* `dmat` is a pointer to a `mp_data_matrix` type that represent a data matrix where the output will be stored (it is equivalent to the data type explained on 8.2.4 but it uses `mpfr_t` variables instead of `double`). `fileout` is a pointer to a `FILE` where the output will be written on.

9.5 Options to change the files created with `TSMCodeFiles`

9.5.0.10 *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.

9.5.0.11 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.

9.6 Using LibTIDES without driver

To illustrate the use of TIDES without driver we will write a C function to compute the elliptic integral of the first kind $F(\phi, k)$ with multiple precision arithmetic. To simplify we restrict the code to $\phi \in [0, \pi/2], k \in [0, 1]$.

With the option `Driver -> False` we create only the ODE files

In[21]:=

```
TSMCodeFiles[ellipFODE, "ellipF", Driver->False]
```

Out[21]=

```
Files "ellipF.h", ellipF.c", written on directory
"/....../TIDESExamples/chapter09".
```

The code with this example is in the file `mpellipticF.c` inside the folder `chapter7`. It must be written manually by the user, and it is described as follows:

1. The head `ellipticF` declares the input and output variables. When we use MPFR it is better to declare the output as an argument of the function, instead a return value, then the variable `mpfr_t ellipf` will contain, after we call `ellipticF`, the elliptic function evaluated at the input values `phi, k`.

   ```
   void ellipticF(mpfr_t ellipf,  mpfr_t phi, mpfr_t k)
   {
   ```

2. To declare the precision of the output of the elliptic function we use the precision of the input variables. We work with the small value between the precision of `phi` and `k`.

```
    int pphi, pk, prec;
    pphi = (int)mpfr_get_prec (phi);
    pk = (int)mpfr_get_prec (phi);
    if(pphi > pk) prec = pk;
    else prec = pphi;
```

3. The next is to declare the variables with the desired precision and initialize them with its value

```
    mpfr_t  x, par, tini, tol;
    mpfr_init2(x,     prec);
    mpfr_init2(par,   prec);
    mpfr_init2(tini,  prec);
    mpfr_init2(tol,   prec);
    mpfr_set_str(x,   "0", 10, TIDES_RND);
    mpfr_set_str(tini,"0", 10, TIDES_RND);
    mpfr_set(par, k, TIDES_RND);
```

4. To declare the tolerance of the integrator we use the value 10^{-d}, where d is the number of decimal digits of precision. Then we need to convert the binary precision `prec` into decimal precision `dprec`.

```
    int dprec;
    dprec = floor(prec/3.3219);
    mpfr_set_si(tol, -dprec, TIDES_RND);
    mpfr_exp10(tol,  tol, TIDES_RND);
```

5. We call the integrator by computing the solution only in the last point.

```
    mp_tides_delta(ellipMP, NULL, 1, 1, 0, &x, &par,
                    tini, phi, 1, tol, tol, NULL, NULL);
```

6. The solution is the value of the variable in the last point that is stored in `x`.

```
    mpfr_set(ellipf, x, TIDES_RND);
}
```

The next code is a main program to check the previous function for several values of the variable and the parameter .

```
#include "mpfr.h"
#include "mp_tides.h"
#include "ellipMP.h"

void ellipticF(mpfr_t ellipf,  mpfr_t phi, mpfr_t k);



int main()
{
    int dig = 30;
    set_precision_digits(dig);
    int i, npoints = 5;
    mpfr_t phi, par, dphi, ppar, ellipF;
    mpfr_init2(phi,   TIDES_PREC);
    mpfr_init2(dphi,  TIDES_PREC);
    mpfr_init2(par,   TIDES_PREC);
    mpfr_init2(ppar,  TIDES_PREC);
    mpfr_init2(ellipF,TIDES_PREC);

    mpfr_set_str(dphi,"1.57079632679489661923132169164", 10, TIDES_RND);
    mpfr_div_si(dphi, dphi, npoints, TIDES_RND);

    printf("\nElliptic integral of first kind: \n");
    mpfr_set_str(ppar,"0.1", 10, TIDES_RND);

    for(i = 0; i <= npoints; i++) {
        mpfr_mul_si(phi, dphi, i, TIDES_RND);
        mpfr_mul_si(par, ppar, i, TIDES_RND);
        ellipticF(ellipF, phi, par);
        mpfr_printf("F(%d Pi/2, %.2Rf) = %.29Re\n", i, par, ellipF);
    }
    return 0;
}
```

After compiling and running the program we obtain

```
Elliptic integral of first kind:
F(0 Pi/2, 0.00) = 0.00000000000000000000000000000e+00
```

```
F(1 Pi/2, 0.10) = 3.14209953866789708725800078077e-01
F(2 Pi/2, 0.20) = 6.29856249577305325853287534572e-01
F(3 Pi/2, 0.30) = 9.53290897470820066486310930390e-01
F(4 Pi/2, 0.40) = 1.29826841666599685155151355841e+00
F(5 Pi/2, 0.50) = 1.68575035481259604287120365780e+00
```

Chapter 10

Computing partial derivatives: the Lorenz problem

10.1 The Lorenz problem

The classical Lorenz problem is defined by the ordinary differential equations

$$\dot{x} = a(y - x), \quad \dot{y} = -x\,z + r\,x - y, \quad \dot{z} = x\,y - b\,z. \tag{10.1}$$

where $\boldsymbol{x} = (x, y, z) \in \mathbf{R}^3$, and $a, b, r \in \mathbf{R}$ are the parameters.

In TIDES we declare the Lorenz ODE by writing

In[22]:=

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

10.2 Computing partial derivatives of the solution of the ODE

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 the parameters. To do that we need to declare, on the driver, a line with an array of elements to send to the differential equation.

To create automatically the driver to compute partial derivatives we will use the option `AddPartials` in `TSMCodeFiles`.

10.2.0.12 *Option*: `AddPartials`

The option `AddPartials` has four possible formats

- $\boxed{\texttt{AddPartials-> \{\{u,v,..\}, s\}}}$

- $\boxed{\texttt{AddPartials-> \{\{u,v,..\}, s, Until\}}}$

- $\boxed{\texttt{AddPartials-> \{\{u,v,..\}, s, Only\}}}$

- $\boxed{\texttt{AddPartials-> \{\{u,v,..\}, listOfOrders\}}}$

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 partials 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 partials to compute.

If no third argument appears (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 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`,

- $\boxed{\texttt{AddPartials-> \{\{y,a\}, 2\}}}$ or $\boxed{\texttt{AddPartials-> \{\{y,a\}, 2\}, Until\}}}$ compute
  $$\frac{\partial}{\partial y_0}, \quad \frac{\partial}{\partial a}, \quad \frac{\partial^2}{\partial y_0^2}, \quad \frac{\partial^2}{\partial y_0 \partial a}, \quad \frac{\partial^2}{\partial a^2}.$$

- $\boxed{\texttt{AddPartials-> \{\{y,a\}, 2, Only\}}}$ computes $\dfrac{\partial^2}{\partial y_0^2}, \quad \dfrac{\partial^2}{\partial y_0 \partial a}, \quad \dfrac{\partial^2}{\partial a^2}.$

- $\boxed{\texttt{AddPartials-> \{\{y,a\}, \{\{1,2\},\{2,3\}\}\}}}$ computes $\dfrac{\partial^3}{\partial y_0 \partial a^2}, \quad \dfrac{\partial^5}{\partial y_0^2 \partial a^3}.$

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 of the solution and the partial derivatives of the solution.

10.3 Application to the Lorenz problem

Let us take the Lorenz problem with the initial conditions: $x_0 = 1, y_0 = 1/3, z_0 = 2/3$, and the parameters $a = 10, b = 8/3, r = 27$. We will integrate the problem from $t_0 = 0$ until $t = 5$. and we want the solution only at the initial and the final point.

10.3.1 Case 1

To compute, and to write into a file, the solution together with the partial derivatives of the solution with respect the parameter a until order 3 ($\partial/\partial a$, $\partial^2/\partial a^2$, $\partial^3/\partial a^3$) we write

In[23]:=

```
TSMCodeFiles[lorenz, "lorenzC1",
     InitialConditions -> {1, 1/3, 2/3},
     ParametersValue -> {10, 8/3, 27},
     IntegrationPoints -> {0, 5},
     AddPartials -> {{a}, 3, Until},
     Output -> "lorenzC1.txt"]
```

Out[23]=

```
Files "dr_lorenzC1.c", "lorenzC1.h", lorenzC1.c", written on directory
"/....../TIDESExamples/chapter10".
```

The output file `lorenzC1.txt` contains two lines of numbers. The first line corresponds to the initial time $t = 0$, and the second one corresponds to the end time $t = 5$. Each row has 13 columns. The first column of the row i is the time t_i, columns 2,3,4 contain $(x(t_i), y(t_i), z(t_i))$, columns 5,6,7 are ($\partial x(t_i)/\partial a, \partial y(t_i)/\partial a, \partial z(t_i)/\partial a)$, columns 8,9,10 are $(\partial^2 x(t_i)/\partial a^2, \partial^2 y(t_i)/\partial a^2, \partial^2 z(t_i)/\partial a^2)$ and columns 11,12,13 are ($\partial^3 x(t_i)/\partial a^3$, $\partial^3 y(t_i)/\partial a^3$, $\partial^3 z(t_i)/\partial a^3$). The number and order of the rows and columns for different outputs (screen or data matrix) is exactly the same.

If we need partial derivatives with respect to only one initial condition or parameter it is relatively easy to find the column that corresponds to each element of the output. It always follows the same order: time, variables and partial derivatives of the variables.

10.3.2 Case 2

Let us take now two more difficult examples. With the same initial conditions and parameters we compute all the partial derivatives, with respect to the initial conditions x_0, y_0 and with respect to the parameter a, until order two. The MathTIDES expression is equal to the previous case but now we change the option `AddPartials`

In[24]:=

```
TSMCodeFiles[lorenz, "lorenzC2",
     InitialConditions -> {1, 1/3, 2/3},
     ParametersValue -> {10, 8/3, 27},
     IntegrationPoints -> {0, 5},
     AddPartials -> {{x, y, a}, 2, Until},
     Output -> "lorenzC2.txt"]
```

Out[24]=

```
Files "dr_lorenzC2.c", "lorenzC2.h", lorenzC2.c", written on directory
"/....../TIDESExamples/chapter10".
```

In this case we compute the 9 partial derivatives (in a different order):

$$\begin{array}{lll} \partial/\partial x_0, & \partial/\partial y_0, & \partial/\partial a, \\ \partial^2/\partial x_0\partial y_0, & \partial^2/\partial x_0\partial a, & \partial^2/\partial y_0\partial a, \\ \partial^2/\partial x_0^2, & \partial^2/\partial y_0^2, & \partial^2/\partial a^2. \end{array}$$

Then the output has 31 columns: the columns of the time, three columns for the variables and 9 partial derivatives for each variable.

10.3.3 Case 3

The next example combines the computation of an extra function together with the computation of partial derivatives

In[25]:=

```
TSMCodeFiles[lorenz, "lorenzC3",
     InitialConditions -> {1, 1/3, 2/3},
     ParametersValue -> {10, 8/3, 27},
     IntegrationPoints -> {0, 5},
     AddPartials -> {{x, y, a}, 2, Only},
     AddFunctions -> {(x - 1)^2 + (y - 1/3)^2 + (z - 2/3)^2},
     Output -> "lorenzC3.txt"]
```

Out[25]=

```
Files "dr_lorenzC3.c", "lorenzC3.h", lorenzC3.c",written on directory
"/....../TIDESExamples/chapter10".
```

It computes the solution and the evolution of the function $D = (x-1)^2 + (y-1/3)^2 + (z-2/3)^2$. This function represents the square of the distance of the solution $\boldsymbol{x}$ to the initial point. This is done by adding the option AddFunctions. Simultaneously we compute the partial derivatives with respect to x_0, y_0 and a only at order two, i.e. the partial derivatives (in a different order):

$$\begin{array}{lll} \partial^2/\partial x_0 \partial y_0, & \partial^2/\partial x_0 \partial a, & \partial^2/\partial y_0 \partial a, \\ \partial^2/\partial x_0^2, & \partial^2/\partial y_0^2, & \partial^2/\partial a^2. \end{array}$$

The partial derivatives of the function D are also calculated.

10.4 Changes in the driver to compute partial derivatives

Reading the ODE files obtained in cases 1 (10.3.1)and 2 (10.3.2) we observe that both files are identical except for the name used. This is not true when we compute an extra function.

The drivers are identical to that obtained without the option AddPartial but they have two more lines.

- A line after including the header files and before the beginning of the main program with the initialization of an array of integers that contains the information that LibTIDES needs to compute the partial derivatives:

```
int   lorenzC1_PDData[]  = {1, 4, 4, 5, 0, 1, 3, 6, 10, 10, 1, 1,
1, 1, 2, 1, 1, 3, 3, 1, 10, 0, 0, 1, 0, 1, 2, 0, 1, 2, 3, 10, 0,
1, 0, 2, 1, 0, 3, 2, 1, 0, 5, 0, 1, 2, 4, 7, 7, 1, 1, 1, 1, 1, 2,
1, 7, 0, 0, 0, 1, 0, 1, 2, 7, 0, 1, 2, 1, 3, 2, 1, 0, 1, 2, 3};
```

  in case 1 and

```
int   lorenzC2_PDData[]  = {3, 1, 2, 4, 10, 11, 0, 1, 3, 5, 7, 10,
14, 18, 21, 25, 28, 28, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 2, 1, 1, 1, 1, 1,1, 2, 1, 28, 0, 0, 1, 0, 2, 0, 3, 0, 1,
4, 0, 2, 1, 5, 0, 3, 1, 6, 0, 2, 7,  0, 3, 2, 8, 0, 3, 9, 28, 0, 1,
0, 2, 0, 3, 0, 4, 1, 0, 5, 1, 2, 0, 6, 1, 3, 0, 7, 2, 0, 8, 2, 3, 0,
9, 3, 0, 11, 0, 1, 2, 3, 4, 6, 8, 10, 12, 14, 16, 16, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1,1, 1, 1, 1, 1, 1, 16, 0, 0, 0, 0, 0, 1, 0, 2, 0, 3, 0,
2, 0, 3, 0, 3, 16, 0, 1, 2, 3, 4, 1, 5, 1, 6, 1, 7, 2, 8, 2, 9, 3,
0, 0, 0, 1, 0, 0, 0,1, 0, 0, 0, 1, 2, 0, 0, 1, 1,0, 1, 0, 1, 0, 2,
0, 0, 1, 1, 0, 0, 2};
```

in case 2.

- The second argument when we call to the integrator is the name of the previous array

```
        dp_tides_delta(lorenzC1, lorenzC1_PDData,
                     nvar, npar, nfun, v, p, tini, dt, nipt,
                     tolrel, tolabs, NULL, fd);
```

in case 1 and

```
        dp_tides_delta(lorenzC2, lorenzC2_PDData,
                     nvar, npar, nfun, v, p, tini, dt, nipt,
                     tolrel, tolabs, NULL, fd);
```

in case 2.

10.5 MathTIDES function `PartialDerivativesText`

Another way to construct the driver to compute partial derivatives is by creating the integrator without using the option `AddPartials`, and changing the driver manually. To do that we need to use the MathTIDES function `PartialDerivativesText`. The expression `PartialDerivativesText` 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 used to declare 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.

The alternative to write the integrators in cases 1 and 2 is to create the files to compute the case without partial derivatives

In[26]:=

```
TSMCodeFiles[lorenz, "lorenzC",
     InitialConditions -> {1, 1/3, 2/3},
     ParametersValue -> {10, 8/3, 27},
     IntegrationPoints -> {0, 5},
     Output -> "lorenzC.txt"]
```

Out[26]=

```
Files "dr_lorenzC.c", "lorenzC.h", lorenzC.c", written on directory
"/....../TIDESExamples/chapter10".
```

After that we use MathTIDES to create the text to initialize the arrays

In[27]:=

```
PartialDerivativesText[{x, y, z}, {a, b, r}, {{a}, 3, Until},"lorenzC1"]
```

Out[27]=

```
int   lorenzC1_PDData[]  = {1, 4, 4, 5, 0, 1, 3, 6, 10, 10, 1, 1, 1,
1, 2, 1, 1, 3, 3, 1, 10, 0, 0, 1, 0, 1, 2, 0, 1, 2, 3, 10, 0, 1, 0,
2, 1, 0, 3, 2, 1, 0, 5, 0, 1, 2, 4, 7, 7, 1, 1, 1, 1, 1, 2, 1, 7, 0,
0, 0, 1, 0, 1, 2, 7, 0, 1, 2, 1, 3, 2, 1, 0, 1, 2, 3};
```

In[28]:=

```
PartialDerivativesText[{x, y, z}, {a, b, r}, {{x, y, a}, 2, Until},
                         "lorenzC2" ]
```

Out[28]=

```
int   lorenzC2_PDData[]  = {3, 1, 2, 4, 10, 11, 0, 1, 3, 5, 7, 10,
14, 18, 21, 25, 28, 28, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 28, 0, 0, 1, 0, 2, 0, 3, 0, 1,
4, 0, 2, 1, 5, 0, 3, 1, 6, 0, 2, 7, 0, 3, 2, 8, 0, 3, 9, 28, 0, 1, 0,
2, 0, 3, 0, 4, 1, 0, 5, 1, 2, 0, 6, 1, 3, 0, 7, 2, 0, 8, 2, 3, 0, 9,
3, 0, 11, 0, 1, 2, 3, 4, 6, 8, 10, 12, 14, 16, 16, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 16, 0, 0, 0, 0, 0, 1, 0, 2, 0, 3, 0, 2,
0, 3, 0, 3, 16, 0, 1, 2, 3, 4, 1, 5, 1, 6, 1, 7, 2, 8, 2, 9, 3, 0, 0,
0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 2, 0, 0, 1, 1, 0, 1, 0, 1, 0, 2, 0, 0,
1, 1, 0, 0, 2};
```

Finally we make a copy and paste to include the line of each case into the driver and we also change the second argument when we call the integrator by the name of the corresponding array.

10.6 Computing the position of each element of the output

In the previous cases 2 and 3 it is difficult to know what is the position of a particular partial derivative at the output. 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.

In the Lorenz problem we have three variables x, y, z and three parameters a, b, r. In case 10.3.1 we compute all the partial derivatives with respect to the parameter a until order 3, then

- `position_variable(0, "2", 3, 0, lorenzC1_PDData)` returns 7. It means that the term $\partial^2 x/\partial a^2$ appears at the column 8.
- `position_variable(2, "0", 3, 0, lorenzC1_PDData)` returns 3. It means that the term z appears at the column 4.
- `position_variable(1, "5", 3, 0, lorenzC1_PDData)` returns -1 because $\partial^5 y/\partial a^5$ is not computed.
- `position_variable(1, "1/1", 3, 0, lorenzC1_PDData)` returns -1 because we only differentiate with respect to one variable.
- `position_function(0, "1", 3, 0, lorenzC1_PDData)` returns -1 because we do not compute any extra function.

In case 10.3.2 we compute the partial derivatives until order 3 with respect to x_0, y_0, a, in this order.

- `position_variable(0, "1/0/1", 3, 0, lorenzC2_PDData)` returns 19. It means that the term $\partial^2 x/\partial x_0 \partial a$ appears at the column 20.
- `position_variable(2, "2/0/0", 3, 0, lorenzC2_PDData)` returns 15. It means that the term $\partial^2 z/\partial x_0^2$ appears at the column 16.
- `position_variable(0, "0/1/0", 3, 0, lorenzC2_PDData)` returns 7. It means that the term $\partial x/\partial y_0$ appears at the column 8.
- `position_variable(1, "0/0/0", 3, 0, lorenzC2_PDData)` returns 2. It means that the term y appears at the column 3.
- `position_variable(1, "5", 3, 0, lorenzC2_PDData)` returns -1 because we compute partial derivatives with respect to three variables.
- `position_variable(1, "1/0/1/0", 3, 0, lorenzC2_PDData)` returns -1 because we compute partial derivatives with respect to three variables.
- `position_function(0, "1/0/1", 3, 0, lorenzC2_PDData)` returns -1 because we do not compute any extra function.

In case 10.3.3 we compute one extra function D, and the partial derivatives, only at order 2, with respect to x_0, y_0, a, in this order.

- `position_variable(0, "1/0/1", 3, 1, lorenzC3_PDData)` returns 13. It means that the term $\partial^2 x/\partial x_0 \partial a$ appears at the column 14.
- `position_variable(2, "2/0/0", 3, 1, lorenzC3_PDData)` returns 7. It means that the term $\partial^2 z/\partial x_0^2$ appears at the column 8.
- `position_variable(0, "0/1/0", 3, 1, lorenzC3_PDData)` returns -1 because we only compute derivatives of order 2.
- `position_variable(1, "0/0/0", 3, 1, lorenzC3_PDData)` returns 2. It means that the term y appears at the column 3.
- `position_function(0, "0/1/1", 3, 1, lorenzC3_PDData)` returns 24. It means that the term $\partial^2 D/\partial y_0 \partial a$ appears at the column 25.
- `position_variable(1, "5", 3, 1, lorenzC3_PDData)` returns -1 because we compute partial derivatives with respect to three variables.
- `position_function(1, "1/0/1", 3, 1, lorenzC3_PDData)` returns -1 because we do compute only one extra function.

Chapter 11

Computing events

11.1 Events

Let's suppose the ODE system

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

and $\boldsymbol{y}(t)$ the solution of this ODE.

Sometimes we want to locate, during the integration of the ODE, some events, like zeros or extrema, of the real function $G(\boldsymbol{y}(t)) : [t_o, t_f] \in \mathbb{R} \to \mathbb{R}$ inside a time interval.

TIDES has the possibility to locate four kind of events: zeros, local extrema, local maxima and local minima. *In all cases, TIDES computes the power series of the function $G(\boldsymbol{y}(t))$ and supposes that there is no more than one zero or extrema inside the convergence interval of this series.* The extrema are located by computing the zeros of the derivative. The zeros are located by a change of sign into the convergence interval. *If there are more then one zero (extrema) inside the interval or the zero corresponds to a multiple zero, our method does not guarantee that we find the event.*

To illustrate how to search events let's take again the sin and cosine differential equation given in (6.1)

$$\dot{x} = y, \quad \dot{y} = -x, \quad x(0) = 0, \quad y(0) = 1,$$

whose solution are the functions $x(t) = \sin t$, $y(t) = \cos t$. We will use this ODE to compute

- **Case 1**: all the zeros of the function $x(t) + y(t) = \sin t + \cos t$ between 0 and 10π.
- **Case 2**: the two first local extrema of the function $y(t) = \cos t$ between 0 and 10π.
- **Case 3**: Ten local maxima of the function $x(t) + 2\,y(t) = \sin t + 2\cos t$ between 0 and 10π.

11.2 Events and MathTIDES

MathTIDES writes the code to compute events by using the expression `TSMCodeFiles` in a similar way than the TSM Integrator is written.

The arguments of `TSMCodeFiles` are the same described in (5.5), but we change the code by means of the options `FindZeros`, `FindExtrema`, `FindMinima`, `FindMaxima`, `EventTolerance` and `EventsNumber`.

The following `TSMCodeFiles` options can not be used to compute events: `MinTIDES`, `RelativeTolerance`, `AbsoluteTolerance`, `AddFunctions` and `AddPartials`. The rest of the options can be used but sometimes they act in a different way.

11.2.1 Event options of `TSMCodeFiles`

11.2.1.13 *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 $G(\boldsymbol{y})$.

11.2.1.14 *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 $G(\boldsymbol{y})$.

11.2.1.15 *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 $G(\boldsymbol{y})$.

11.2.1.16 *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 $G(\boldsymbol{y})$.

11.2.1.17 *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]`.

11.2.1.18 *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.

11.2.2 Changes in old options of `TSMCodeFiles`

11.2.2.19 *Option*: `IntegrationPoints`

The use of the option `IntegrationPoints` is similar than before, but the result is slightly different. In fact only the initial and the final integration points are considered.

11.2.2.20 *Option*: `DataMatrix`

The option `DataMatrix` is similar than before, but when it is used to compute events we have two differences

- All the computed events are stored in the data matrix. The number of rows of the data matrix is equal to the number of events.
- If we use the option `DataMatrix -> True` the name of the data matrix where the events are stored is the name of the file joined to `_EventsVector`.

11.2.3 Case 1

The sine and cosine differential equation `sincosODE` has been declared in MathTIDES as in the section (6.2). Then the ODE files and the driver to compute the events of the first case are obtained with the expression

In[29]:=

```
TSMCodeFiles[sincosODE,
        "sincosMinZ",
        InitialConditions -> {0, 1},
        IntegrationPoints -> {0, 10 Pi},
        Output -> Screen,
        FindZeros -> x + y];
```

The output of this driver is

```
2.356194490192345e+00   7.071067811865476e-01  -7.071067811865476e-01   0.000000000000000e+00
5.497787143782138e+00  -7.071067811865476e-01   7.071067811865475e-01   0.000000000000000e+00
```

```
8.639379797371932e+00   7.071067811865472e-01  -7.071067811865471e-01   0.000000000000000e+00
1.178097245096173e+01  -7.071067811865474e-01   7.071067811865469e-01  -2.220446049250313e-16
1.492256510455152e+01   7.071067811865470e-01  -7.071067811865471e-01   0.000000000000000e+00
1.806415775814131e+01  -7.071067811865469e-01   7.071067811865469e-01  -2.775557561562891e-17
2.120575041173110e+01   7.071067811865470e-01  -7.071067811865468e-01   0.000000000000000e+00
2.434734306532090e+01  -7.071067811865468e-01   7.071067811865468e-01   0.000000000000000e+00
2.748893571891069e+01   7.071067811865470e-01  -7.071067811865469e-01   0.000000000000000e+00
3.063052837250048e+01  -7.071067811865470e-01   7.071067811865470e-01   0.000000000000000e+00
```

Each line of this output represents an event. The first column is the time t where the event occurs. The columns 2 and 3 represents the solution x, y in this point. Finally the last column represent the value of the event function (zero in this case).

The position of the elements in output is the same for a file and for a data matrix.

11.2.4 Case 2

In this case we compute the two first extrema of the cosine function

```
In[30]:=

TSMCodeFiles[sincosODE,
        "sincosMinE",
        InitialConditions -> {0, 1},
        IntegrationPoints -> {0, 10 Pi},
        Output -> Screen,
        FindExtrema -> x,
        EventsNumber -> 2];
```

and we obtain the following result

```
1.570796326794897e+00   1.000000000000000e+00   0.000000000000000e+00   1.000000000000000e+00
4.712388980384690e+00  -1.000000000000000e+00   0.000000000000000e+00  -1.000000000000000e+00
```

We see in the previous lines that the first value corresponds to a local maximum and the second point is a minimum. When the integrator detects the second event it stops.

11.2.5 Case 3

In this case we try to find 10 zeros of the functions $\sin t + 2\cos t$ in $[0, 10\pi]$.

```
In[31]:=
TSMCodeFiles[sincosODE,
        "sincosMinM",
        InitialConditions -> {0, 1},
        IntegrationPoints -> {0, 10 Pi},
        Output -> Screen,
        FindMaxima -> x + 2 y,
        EventsNumber -> 10];
```

The output

```
4.636476090008061e-01   4.472135954999579e-01   8.944271909999159e-01   2.236067977499790e+00
6.746832916180392e+00   4.472135954999577e-01   8.944271909999157e-01   2.236067977499789e+00
1.303001822335998e+01   4.472135954999574e-01   8.944271909999155e-01   2.236067977499788e+00
1.931320353053956e+01   4.472135954999575e-01   8.944271909999151e-01   2.236067977499788e+00
2.559638883771915e+01   4.472135954999578e-01   8.944271909999147e-01   2.236067977499787e+00
```

has only five lines because the funcion $\sin t + 2\cos t$ has only five maxima in this interval.

11.3 Events and LibTIDES

If we observe the driver `dr_sincosMinZ` we find two differences with respect to a TSM integrator driver. The line

```
int nevents = 0;
```

declares an integer variable that contains the maximum number of events we may compute. In this case the value 0 indicates that we want to compute all the events inside the integrator interval.

The line

```
dp_tides_find_zeros(sincosMinZ, nvar, npar, v, NULL,
                     tini, tend, tol, &nevents, NULL, fd);
```

substitutes the integrator `dp_tides_delta` or `dp_tides_list` by the event generator, in this case to find zeros. Substitute the word `zeros` by `extrema`, `minima`, `maxima` to find other events.

11.3.1 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 ODE function. 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 searchs 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 searchs 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.

11.4 Finding the period of a periodic orbit

Another use of the events seeker is to find the period of a periodic orbit. Let's take the Kepler problem of section 7.1 with the same initial conditions $\boldsymbol{x} = (0.8, 0, 0), \boldsymbol{X} = (0, 1.2247448713915892, 0)$ and value of the parameter $\mu = 1$. With these values the orbit is periodic of period 2π.

The period T of a periodic orbit is a value T such us $\boldsymbol{x}(T) - \boldsymbol{x}_0 = 0$. Find this value is the same that find a zero of the value of the square[1] of the distance between the solution $\boldsymbol{x}$ and the initial value $\boldsymbol{x}_0$, in this case $d^2 = (x - 0.8)^2 + y^2 + z^2$.

The MathTIDES expression to write the ODE functions and the driver is

```
In[32]:=

TSMCodeFiles[keplerODE,
        "keplerP",
        InitialConditions -> {0.8, 0, 0, 0, 1.2247448713915892, 0},
        ParametersValue -> {1},
        IntegrationPoints -> {0, 10 Pi},
        EventsNumber -> 2,
        FindZeros -> (x - 0.8)^2 + y^2 + z^2,
        Output -> Screen];
```

Let us observe that we try to compute two events. This is because the first value where the distance is zero is the initial point, then the period appears as the second zero of this function. The (simplified) output in this case is

```
0.000000000000e+00   8.00e-01   0.00e+00   0.00e+00   0.00e+00   1.22e+00   0.00e+00   0.00e+00
6.283185307179e+00   8.00e-01   0.00e+00   0.00e+00   3.88e-16   1.22e+00   0.00e+00   0.00e+00
```

[1]We do not use the square root because the information is the same, we need to make more operations and finally the series of the square root has a singular value in zero.