Software for the numerical integration of ODE by means of high-order Taylor methods (II)

Àngel Jorba
angel@maia.ub.es

University of Barcelona

Advanced Course on Long Term Integrations

Outline

The software can be retrieved from
`http://www.maia.ub.es/~angel/taylor/`

It installs in a GNU/Linux system.

It requires the packages `flex` and `bison`.

Now we will see how it works.

```
/* ODE specification: rtbp */
mu=0.01;
umu=1-mu;
r2=x1*x1+x2*x2+x3*x3;
rpe2=r2-2*mu*x1+mu*mu;
rpe3i=rpe2^(-3./2);
rpm2=r2+2*(1-mu)*x1+(1-mu)*(1-mu);
rpm3i=rpm2^(-3./2);

diff(x1, t)= x4+x2;
diff(x2, t)= x5-x1;
diff(x3, t)= x6;
diff(x4, t)= x5-(x1-mu)*(umu*rpe3i)-(x1+umu)*(mu*rpm3i);
diff(x5, t)=-x4-x2*(umu*rpe3i+mu*rpm3i);
diff(x6, t)=-x3*(umu*rpe3i+mu*rpm3i);
```

To produce a numerical integrator for this vector field, assume that the previous code is in the file rtbp.in
Then, you can type

```
taylor -name rtbp -o taylor_rtbp.c -step -jet -sqrt rtbp.in
taylor -name rtbp -o taylor.h -header rtbp.in
```

to produce two files:

- taylor_rtbp.c: The time stepper
- taylor.h: Header to define the arithmetic used

```
Usage: ./taylor
 [-name ODE_NAME]
 [-o outfile]
 [-doubledouble | -qd_real | -dd_real | -gmp -gmp_precision PRECISION]
 [-main | -header | -jet | -main_only]
 [-step STEP_CONTROL_METHOD]
 [-u | -userdefined] STEP_SIZE_FUNCTION_NAME ORDER_FUNCTION_NAME
 [-f77]
 [-sqrt]
 [-headername HEADER_FILE_NAME]
 [-debug] [-help] [-v]  file
```

Main C call:

```
int taylor_step_ODE_NAME(MY_FLOAT *time,
                         MY_FLOAT *xvars,
                         int      direction,
                         int      step_ctrl_method,
                         double   log10abserr,
                         double   log10relerr,
                         MY_FLOAT *endtime,
                         MY_FLOAT *stepused,
                         int      *order)
```

Main Fortran 77 call:

```
void taylor_f77_ODE_NAME__(MY_FLOAT *time,
                           MY_FLOAT *xvars,
                           int      *direction,
                           int      *step_ctrl_method,
                           double   *log10abserr,
                           double   *log10relerr,
                           MY_FLOAT *endtime,
                           MY_FLOAT *stepused,
                           int      *order,
                           int      *flag)
```

Let us see an example of numerical integration by selecting the initial condition `x1=-0.45`, `x2=0.80`, `x3=0.00`, `x4=-0.80`, `x5=-0.45` and `x6=0.58`.

We will perform a numerical integration with the standard double precision of the computer, for 1 unit of time.

As a first test, we will check the preservation of the Hamiltonian.

We have coded a small main program that uses this initial condition to call the Taylor integrator till the time has advanced in one unit.

Next run takes $\varepsilon_a = \varepsilon_r = 10^{-16}$

```
value of H at the initial condition:  -1.3362071584596453
numerical integration starts...
    0.24011923241902   20  -1.00000
    0.49521588761001   20   0.00000
    0.76536594703474   20   0.00000
    1.00000000000000   20  -1.00000
```

It is also interesting to run the program for a longer time span.

It is possible to check (statistically) that the variation of the energy behaves like a random walk.

Let H_j be the value of H at the step number j of the numerical integration and, instead of consider $H_j - H_0$, let us focus on the local variation $H_j - H_{j-1}$.

	$\varepsilon = 10^{-14}$	$\varepsilon = 10^{-15}$	$\varepsilon = 10^{-16}$	$\varepsilon = 10^{-17}$	$\varepsilon = 10^{-18}$
-4	0	0	0	0	0
-3	45	2	7	5	6
-2	32,904	21,155	21,377	21,372	21,662
-1	772,723	745,668	760,755	768,334	777,760
0	1,970,571	2,084,758	2,134,729	2,157,287	2,174,276
1	765,519	744,438	760,183	767,596	776,776
2	32,444	21,174	21,576	21,696	21,949
3	42	6	5	3	5
4	0	0	0	0	0

Local variation of the energy during 10^6 units of time. The first column denotes multiples of the machine precision and the remaining columns contain the number of integration steps for which the local variation of energy is equal to the multiple of eps in the first column.

To do an standard statistical analysis, let us assume that the sequence of errors $H_j - H_{j-1}$ is given by a sequence of independent, identically distributed random variables, and we are interested in knowing if its mean value is zero or not.

Therefore, we will apply the following test of significance of the mean. The null hypothesis assumes that the true mean is equal to zero.

If k denotes a multiple of eps and ν_k the number of times that this deviation has occurred (in our case, $\nu_k = 0$ if $k > 4$), we define

$$n = \sum_{|k|\leq 4} \nu_k, \qquad m = \frac{1}{n} \sum_{|k|\leq 4} k\nu_k, \qquad s = \sqrt{\frac{1}{n^2} \sum_{|k|\leq 4} (k-m)^2 \nu_k},$$

where s stands for the standard error of the sample mean m.

Under the previous assumptions (independence and equidistribution of the observations), the value

$$\tau = \frac{m}{s}$$

must behave as a $N(0,1)$ standard normal distribution.

To test the null hypothesis (i.e., zero mean) with a confidence level of 95%, we have to check for the condition $|\tau| \leq 1.96$.

	$\varepsilon = 10^{-14}$	$\varepsilon = 10^{-15}$	$\varepsilon = 10^{-16}$	$\varepsilon = 10^{-17}$	$\varepsilon = 10^{-18}$
-4	0	0	0	0	0
-3	45	2	7	5	6
-2	32,904	21,155	21,377	21,372	21,662
-1	772,723	745,668	760,755	768,334	777,760
0	1,970,571	2,084,758	2,134,729	2,157,287	2,174,276
1	765,519	744,438	760,183	767,596	776,776
2	32,444	21,174	21,576	21,696	21,949
3	42	6	5	3	5
4	0	0	0	0	0
τ	-6.0613	-0.9160	-0.1383	-0.0735	-0.3141

The last row shows the value of τ for the different integrations. It is clear that for $\varepsilon = 10^{-14}$ we must reject that the drift has zero mean, and it is also clear that this hypothesis cannot be rejected in the other cases.

A comparison with a Runge-Kutta-Fehlberg 7-8

We ask the rk78 for an accuracy of 10^{-16}.

The error in H after 10^6 units of time, in number of multiples of the machine epsilon, is -13412 for the rk78, and -217 for the Taylor method.

The time taken for the rk78 was of 9m and 34s, while the Taylor method needed 4m and 4s.

If we ask the rk78 for an accuracy of 10^{-14} then the time taken goes down to 4m and 58s, but the final error is 649368 times the epsilon of the machine (that is, 1.44×10^{-10}).

This is a second benchmark using the standard quadruple precision of a HP 9000/712 computer, with a 100 MHz PA-RISC 1.1 processor.

We have used the vector field of the Restricted Three-Body Problem, with the same initial condition and mass parameter as before.

The integration time has been restricted to 10 units, to avoid long testing times. We have asked for a local error of 10^{-32} for the rk78, and of the 10^{-33} for the Taylor method.

The total cpu time for the rk78 is of 3m 48s, while the Taylor method only takes 4.1 seconds.

Next, we will discuss the capabilities of `taylor` to use different arithmetics.

When `taylor` generates the code for the jet of derivatives and/or the step size control, it declares all the real variables with a special type called `MY_FLOAT`, and each mathematical operation is substituted by a suitable macro call (the name of these macros is independent from the arithmetic).

The definition of the type `MY_FLOAT` and the body of the macros is contained in a header file. This file is produced invoking `taylor` with the flag `-header` plus a flag specifying the arithmetic wanted. For instance, to multiply two real numbers ($z = xy$), `taylor` outputs the code

```
  MultiplyMyFloatA(z,x,y);
```

If we call taylor with the -header flag and without specifying the desired arithmetic, it will assume we want the standard double precision and it will generate a header file with the lines,

```
typedef double MY_FLOAT;
```

to define MY_FLOAT as double. We will also find the line

```
/* multiplication r=a*b */
#define   MultiplyMyFloatA(r,a,b)   (r=(a)*(b))
```

If we use the flag -gmp to ask for the GNU multiple precision arithmetic (see below), we will get

```
#define MY_FLOAT  mpf_t
```

and

```
/* multiplication r=a*b */
#define   MultiplyMyFloatA(r,a,b)   mpf_mul(r,(a),(b))
```

Here, `mpf_mul` is the gmp function that multiplies the two numbers `a` and `b` and stores the result in `r`. Then, the C preprocessor will substitute the macros by the corresponding calls to the arithmetic library.

The package includes support for several extended precision arithmetics: DOUBLEDOUBLE, DD_REAL, DQ_REAL and GMP (the GNU Multiple Precision Library).

If a library does not contain implementation of trigonometric functions and/or transcendental functions, we note that they can be defined by means of differential equations. Therefore, if an ODE includes some of these functions, we can enlarge the system of ODEs by adding the differential equation for the special function and to integrate the whole system.

None of these floating point libraries is included in our package. They can be downloaded from the internet and are only needed if extended precision is required.

Note that to use an arithmetic different from the ones provided here we only have to modify the header file (for more details, see the manual...).

Next, we are going to use extended precision (more concretely, the `gmp` library) to compute the error of the double precision version.

To measure the error, we have computed the relative difference between these two approximations. For instance, for the x coordinate, the operations we have implemented are,

$$e(x) = 1 - \frac{\tilde{x}}{x},$$

where x is the extended precision approximation and $\tilde{x}$ is the double precision result. All the computations have been done in double precision. The result is written in multiples of the machine precision.

```
value of H at the initial condition: -0.1336207158e1
numerical integration starts...
1  0.183827140545086 174 -0.82041748043202e-153
2  0.374344795509428 174 -0.59666725849601e-153
3  0.574965855180706 174 -0.67125066580801e-153
4  0.789299521404807 174 -0.14916681462400e-153
5  1.000000000000000 174 -0.22375022193600e-153
iterates: 5  final time: 1.000000e+00
```

Numerical integration using gmp with a 512 bits mantissa and asking for a relative error of 10^{-150}.

```
value of H at the initial condition: -0.1336207158e1
numerical integration starts...
1  0.180071007388544 346 0.124236998889749e-303
2  0.366492191198110 346 0.678258138919457e-303
3  0.562476214016376 346 0.878726168201663e-303
4  0.771527181403796 346 0.921848099579533e-303
5  0.997166681972274 346 0.106043126217200e-302
6  1.000000000000000 346 0.106043682485665e-302
iterates: 6  final time: 1.000000e+00
```

As before, but using a 1024 bits mantissa and asking for a relative error of 10^{-300}.