The Taylor Center for PCs: Exploring, Graphing and Integrating ODEs with the Ultimate Accuracy

Alexander Gofen

The Smith-Kettlewell Eye Research Institute, 2318 Fillmore St., San Francisco, CA 94102, USA
galex@ski.org, www.ski.org/gofen

Abstract. A software package intended as a cross-platform integrated environment for exploring, graphing, real time 2D and 3D "playing" and integrating ODEs with the Modern Taylor method, is presented. Designed under Windows in Delphi-6, the package implements a powerful Graphical User Interface (GUI), and is portable to windowed versions of Linux also. The downloadable Demo of the package displays a variety of meaningful examples including such an illustrative as the Problem of Three Bodies and "Choreography" [8].

1 Introduction

It seems, there were no sophisticated Taylor Solvers designed for PCs since ATOMFT [2], and definitely not as a cross platform Windows/Linux application developed in Delphi. Thus, the goal was to develop a new version of such a package, called the Taylor Center (similarly to "Audio Centers" with radio, cassette, CD players). It has to take full advantage of:

(1) Hardware breakthroughs such as linear memory and the high speed/memory characteristics of the modern PCs;

(2) Software breakthroughs such as the new metaphor of modern operating environments – Graphical User Interface (GUI) and event driven logic, supported by such developer-friendly and efficient systems as Delphi-6 (for Windows) and Kylex (for Linux);

(3) Some new features of the Modern Taylor method and its applications.

From the programmatic point of view, any Taylor solver differs from conventional integrators in that the input – an Initial Value Problem – should be provided not as a function simply computing the right hand parts, but rather as arithmetic expressions representing the right hand parts themselves in order to enable *automatic differentiation*. Typically, these right hand parts must be (automatically) reduced to the so called canonic equations first [1, 3], but this version of the Taylor solver is built around an algorithm evaluating postfix sequences obtained while parsing the so called Linear Elaborated Declarations (in ADA terminology):

$$\begin{aligned}&u_0 = f_0(\{numeric\ values\ only\});\\&u_1 = f_1(u_0, \{numeric\ values\});\\&\ldots\\&u_n = f_n(u_0, u_1, \ldots, u_{n-1}, \{numeric\ values\}).\end{aligned} \qquad (\mathbf{1})$$

Here each variable first appears in the left hand part before it may be used in the right hand parts of the subsequent equations (just like the principle of declaring constants in Pascal).

To evaluate (1), first we apply a recursive parsing procedure to each line to obtain its postfix sequence, whose operands are either numeric values, or the previously computed variables. The evaluation process of the postfix sequence consists of locating triplets (Operand1, Operand2, Operation °) in it, evaluating the elementary equation

Result := Operand1 ° Operand2,

and replacing the triplets with the corresponding Result until the whole sequence reduces to just one term – the final result of the evaluation.

The modern Taylor Method means basically iterative evaluation of recurrent equations similar to (1). Let us consider the standard Initial Value Problem for ODEs:

$$u_k' = f_k(u_1, \ldots, u_m), \quad u_k|_{t=a} = a_k, \qquad k = 1, \ldots m \qquad (\mathbf{2})$$

where u_k' means derivatives with respect to t. The equations (2) make it possible to obtain the set of all first order derivatives u_k' by computing the right hand parts. According to the classical Taylor method, these equations make it possible to obtain the set of all N-order derivatives $u_k^{(N)}$ also. To do that, we have to recurrently apply the well known rules of differentiation to both parts of the equations and substitute the already obtained numerical values. As such, the classical Taylor method had not been considered as a numerical method for computers at least for two reasons: applying the formulas for *N-th* derivative to complex expressions (superposition of several functions in the right hand parts) is both challenging and computationally wasteful due to the exponentially growing volume of calculations in the general case).

On the contrary, the Modern Taylor Method capitalizes on the fact that however complicated the right hand parts are, they are equivalent to a sequence of simple formulas (instructions). Indeed, this sequence of instructions is exactly that resulting from the process of *postfix evaluation* (which is similar to reducing the source system to the so called *Canonical* equations [1, 3]). If the set of the allowed operations consists of the four arithmetic operations (and also raising to constant power, exponent and logarithm), computing the *N-th* derivative requires only no more than $O(N^2)$ operations and it is easily programmable [6]. Thus, unlike its classical counterpart, the Modern Taylor Method becomes viable and competitive for a computer implementation.

2 How it is implemented

To achieve more flexibility in specifying an Initial Value problem (2), the source data is organized as *Linear Elaborated Declarations* in four sections in the given order: Symbolic constants, Initial values, Auxiliary variables, ODEs (Table 1).

Table 1. How the source data of the Initial Value Problem is organized as *Linear Elaborated Declarations*

Variables and equations	Explanation
c1 = NumericValue *c2 =C2(c1, NumericValues)*	*Symbolic constants and functions of them*
u1 = U1(c1, c2,…, NumericValues) . . . . *um = Um(c1, c2,…, u1,…, NumericValues)*	*Initial values for the Main variables and functions of them*
v1 = V1(c1,…, um, NumericValues) *v2 = V2(c1,…, um, v1, NumericValues)*	*Auxiliary variables*
u1' = f1(u1,…, um, c1, c2,…, v1, v2,…) . . . . *um' = fm(u1,…, um, c1, c2,…, v1, v2,…)*	*The source ODEs*
	Internal Auxiliary variables

The Symbolic constants and Auxiliary variables make it possible to parameterize Initial Value problems and to optimize computation by eliminating re-calculations of common sub-expressions. For example, in the Three Body Problem the section of *Constants* is:

```
i15 = -1.5 {exponent index used in subsequent equations}
m1 = 1   {masses}
. . . . . . .
x1c = 1   {initial positions of the three bodies}
. . . . . .
x3c = cos(240)
. . . . . . .
```

The equations in the section *Auxiliary variables* are these:

```
dx12 = x1 - x2            {projections of the distances between the bodies}
. . . . . . . . .
r12 = (dx12^2 + dy12^2)^i15    {common sub-expressions of the ODEs}
r23 = (dx23^2 + dy23^2)^i15
r31 = (dx31^2 + dy31^2)^i15
```

The ODEs themselves are:

```
x1' = vx1
. . . . . . . . .
vy3' = m2*dy23*r23 - m1*dy31*r31
```

Without the auxiliary variables, the more familiar, but non-optimized equations would look like this (just the last one)

$$v_{y3}' = m_2(y_2-y_3)((x_2-x_3)^2 + (y_2-y_3)^2)^{-3/2} + m_1(y_1-y_3)((x_3-x_1)^2 + (y_3-y_1)^2)^{-3/2}$$

(see [7] for the downloadable Demo). The data from the Table 1 is to be processed in the following three stages:

(1) Like the list of linear elaborated declarations, it should be parsed line by line into the postfix sequence for the subsequent automatic differentiation process;

(2) The differentiation process up to the specified order is performed. The set of coefficients of the Taylor expansion (also called the Analytical elements) for each variable is obtained;

(3) With the obtained Taylor coefficients, the convergence radius R should be estimated, an integration step $h<R$ determined and the Taylor expansion applied to increment the all main variables. Then, the stages 2 and 3 may be repeated as many times as necessary (unless a singularity is reached).

The syntax checking and parsing for the stage 1 is done just as in the case of an arbitrary arithmetic expression [4]. As to the evaluation, here we need a more sophisticated algorithm, capable of evaluating N-order derivatives for results of each operation in the postfix sequence. To achieve that, we will "compile" the postfix sequences into the three-address instructions of an abstract Processor, and create the software implementation of that Processor. This "compiled set of instructions" for a given problem would remain the same even if we need to switch from real numbers to complex numbers, or from one real type to another, including the cases of software emulated very long floating point types.

3 The Evaluating and Differentiating Processor

The *memory* for this Processor is the special dynamic array of auxiliary variables AuxVars:

```
AuxVars : array of array of extended;
```

An element *AuxVars[i]* corresponds to the variable or symbolic constant in line *i* of Table 1, while *AuxVars[i, j]* means the *j*-order derivative of that variable (when non-constant).

The format of the instruction of the Processor is this:

```
type TInstruction = record
     ty : (cc, cv, vc, vv); {flags: (C)onstant  or (V)ariable}
     op : char;                        {operation to perform}
     i1, i2, i3 : word    {addresses, i.e. indexes in AuxVars}
     end;
```

For each postfix triplet (*Operand1, Operand2, Operation*) there is an instruction of the type TInstruction, such that *Operand1* is in *AuxVars[i1]*, *Operand2* in *AuxVars[i2]*, and the result of performing of the *Operation* is assigned to *AuxVars[i3]*. To compile a "program" evaluating a postfix sequence, we should scan the postfix sequence from left to right until a first triplet *(Operand1, Operand2, Operation)* is encountered. So, the postfix sequence is encoded in a set of instructions, preserved for the subsequent massive computations of the corresponding arithmetic expression.

Actually, this Processor is designed to do not only simple evaluations like

$$Result := Operand1 \circ Operand2,$$

but also to perform the iterative differentiation up to any given order

$$Result^{(N)} := (Operand1 \circ Operand2)^{(N)}, \quad N = 0, 1, 2,\ldots$$

given the corresponding formulas for computing *N*-th derivative of certain functions – for example the Leibnitz formula

$$(uv)^{(N)} = \sum_{i=0}^{N} u^{(i)} v^{(N-i)}$$

and other similar ones [1,3,6] (all the derivatives are normalized meaning $u^{(i)}/i!$). So, with this Processor we can obtain derivatives of any required order in the initial point *a* for all variables (Table 1) representing the source initial value problem.

4. Estimation of the Convergence Radius

To estimate the convergence radius, we use the algorithm based on the formula of Cauchy-Hadamard

$$R^{-1} = \sup |a_n|^{1/n}$$

where Taylor coefficients a_n are obtained for each unknown function in the source ODEs. Each component of the solution may have its own radius, thus the procedure selects the minimal among them, which should be considered as a convergence radius for the whole system. Although the formula is exact, the result is heuristic because the formula is applied to only a finite number of terms, with simplified assumptions for obtaining the supremum. As a matter of fact, in order to obtain the right integration step *h*, we neither can nor even need to know the exact value of the radius: in any case the step value is going to be adjusted to meet the required accuracy.

So, given the Taylor coefficients $a_0, a_1, ..., a_n$, first of all we obtain their floating point exponents $e_0, e_1, ..., e_n$ using the standard procedure *FrExp(x, m, result)* for all non-zero coefficients, and assigning -4951 (the minimal possible type *extended* exponent) to zero coefficients. These are the rough $\log_2$ of the values, so that logarithms of the sequence of the required fractional powers for further processing are these:

$$e_1/1,\ e_2/2,\ ...,\ e_n/n = s_n.$$

The supremum of the sequence is the greatest point of condensation. To find it for this finite sequence (usually 30-40 terms), the following heuristic procedure is applied.

The sequence $\{s_n\}$ is sorted in descending order (suppose, it is already done). A condensation point of the sequence most probably belongs to such a segment $[s_{i+w}; s_i]$, whose length $s_i - s_{i+w}$ is minimal (*w* is about *n/6*, usually 4-5). Thus, we scan $\{s_n\}$ for the most narrow difference $s_i - s_{i+w}$, $i=1,2,...$ expecting that the greatest condensation point must be there. The *i*, delivering that minimum, is further considered for obtaining the average *s* of s_i ,…, s_{i+w}, and then (returning from the binary logarithms back to the basic values), the value 2^{-s} is accepted as a heuristic radius of convergence *R*. As many numeric experiments showed, this very economical procedure delivers quite a reasonable value for *R*.

5. The Step Size and Accuracy Control

The problem of selecting the optimal order of approximation *N* and integrating step *h* (to meet the required accuracy) is vital for any implementation of the Taylor method [1,3]. For example, according to Moore's rule of thumb, it is advised to set *N* as the number of decimal digits in the floating point mantissa (19 for the 8-byte mantissa in the type *extended*), and to specify *h* so that the quotient $q=h/R=0.1$. In the Taylor Center the defaults for these values are $N=30,\ q=0.5$. This *q* affects each of the three modes of the error control.

In the *No error control* mode the value of *q* is not automatically adjusted and remains that specified by the user. Generally speaking, the smaller the quotient, the higher the accuracy, but more steps would be necessary to cover the given segment (which may compromise the accuracy).

Even with no error control we can estimate the actual accuracy by switching the direction back and reaching the initial point. The column of the differences then will show how far we are away from the initial values.

Another approach to estimate the accuracy in this mode is to perform the integration until the given point twice with different quotients q, say with $q = 0.2$ and $q = 0.05$, and to compare the results.

The following two methods of error control rely on the values of the absolute and/or relative error tolerance entered by the user.

The *Back-step error control* mode changes the algorithm of integration so that each integration step occurs in a loop of step adjustments in the following way.

(1) The step h obtained in point t with the given starting value of $q = q_0$ (shown on the display) is applied, then the algorithm computes the new element in point $t+h$ and the Taylor expansion in this new point with the step value $(-h)$. Ideally that must be equal to the solution in the previous point t.

(2) If the components of the solution fit the specified error tolerance in the table, this step doesn't require any adjustment and it ends the loop. Otherwise…

(3) The step is split (halved), and the process repeats either until the specified accuracy is reached, or the limit (of 30 splitting) is exceeded. The latter means that the specified accuracy cannot be achieved for the given point due to a variety of reasons. (The 30 splits mean that the last attempt was made with the step as little as 2^{-30} of the initial step).

Similarly, the *Half-step error control* also arranges a loop of step adjustments with the goal of achieving the specified accuracy. It works as follows.

(1) The step h obtained with the given starting value of $q = q_0$ is applied, and then the algorithm computes the new element in point $t+h$ to be used as a vector of check values. The step h is split: $h = h1 + h2$.

(2) The values in the same point $t+h$, $h = h1 + h2$, are obtained in two steps and then compared with the check values. If the components of the solution fit the specified error tolerance, this step doesn't require any adjustment and ends the loop. Otherwise…

(3) The step is split (halved), and the process repeats either until the specified accuracy is reached, or the limit (of 30 splitting) is exceeded. The latter means that the specified accuracy cannot be achieved for the given point.

6 Reaching the Ultimate Accuracy

The ultimate accuracy at each *finite* integration step (i.e. the accuracy up to all available digits of the mantissa) is an ambitious, but achievable goal in the framework of the Taylor method and specifically in this package (the user has to specify the relative error tolerance for the desired components say less than 2^{-64}).

Several cases of essentially unstable problems (the two body problem, or the three body Lagrange case producing symmetrical ellipses with low enough eccentricities) were tested with the goal of reaching the ultimate accuracy. About 150-200 steps were required to cover one period, and this highest accuracy was achieved at each step (maybe except one).

The reason while the Taylor method may fail in achieving the highest accuracy is the same that may happen in any numeric computations with floating point numbers: a *subtractive error*. It occurs when a sum or difference of the computed values is zero, or so near zero, that the exponent of the result is N units smaller than either of the operand's exponents (N is near the length of the mantissa, 64 for the type *extended*). Whether the effect of such a dramatic loss of precision is catastrophic for a given computational process depends on the way it propagates farther to the end result.

For the Taylor method, summing is a part of computing almost all derivatives, thus the subtractive error occurs when any of the n-order derivatives or its computed sub-expressions in a current point reaches a near-zero value. The bigger the n, the easier this effect is eliminated by the factor h^n while selecting a smaller step h. In the worst exceptional case when $n=1$ (or the Taylor series' sum is zero), no small h can cure the problem in this point. Still, unless this point is not a singularity for the solution, there is always a simple remedy: we can avoid this exceptional point selecting the previous step a bit smaller. After that, a new finite step would make it posiible to easily jump over this "problem" point (according to the theory of analytical functions, the zero points may be only isolated).

It is worth noting that whichever integrating method is used, the ultimate accuracy at each and every *one* integration step may still not be enough for obtaining all correct digits in the result of the *two or more* integration steps: such is the nature of essentially unstable problems like those mentioned above. For example, for a Lagrange case of the three body problem, in spite of the fact that the accuracy at each individual step was as high as 10^{-19}, after about 150 such steps covering the full period, the accuracy dropped till only 10^{-13}.

Therefore, in order to achieve the best accuracy in multiple step integration, we have to:

(1) Minimize the number of steps needed to cover the required segment by using a method of integration with *finite* steps independent on the required accuracy. Indeed, this method is the Taylor method.

(2) Implement a longer (or of any given length) type of float numbers and arithmetic over them as a software emulation (it exists in several versions of Pascal).

7 Features of the Taylor Center

With this version of the Taylor Center you can:

- Specify and study the Initial Value Problems for virtually any system of ODEs in the standard format with numeric and symbolic constants and parameters;

- Perform numerical integration of Initial Value Problems with the highest possible accuracy while the step of integration remains finite and does not approach zero (instead, the order of approximation is high);

- Perform integration either "blindly", or graphically visualized. It stops either after a given number of steps, or when an independent variable reaches a given value, or when a dependent variable reaches a given value (as explained in the next item);

- Switch integration between several versions of ODEs defining the same trajectory, but with respect to different independent variables. For example, it is possible to switch the integration from that with respect to t to that by x, or by y in order to reach a given end value (or zeros) of a dependent variable;

- Integrate piecewise-analytical ODEs;

- Specify different methods of controlling the accuracy and the step size;

- Specify accuracy for individual components either as an absolute or relative error tolerance, or both;

- Apply the order of approximation as high as you like (30, or 40 or whatever), and get the solution in the form of the set of analytical elements – Taylor expansions covering the required domain;

- Study Taylor expansions and the radius of convergence for the solution at all points of interest up to any high order (with the only limitation that the terms in the series do not exceed the maximum value of about 10^{4932} implied by the 10-byte implementation of the real type extended);

- Graph color curves (trajectories) for any pair of variables of the solution – up to 7 on one screen – either as plane projections, or as 3D stereo images (for triplets of variables) to be watched through anaglyphic (red/blue) glasses;

- "Play" dynamically the near-real time motion along the computed trajectories either as 2D or 3D stereo animation;

- Explore several meaningful examples supplied with the package such as the problem of Two and Three Bodies. Symbolic constants and expressions make it possible to parameterize the equations and initial values and to easily try different initial configurations of special interest.

In particular, the examples include a fascinating case of the so called Choreography for the Three Body motion, an eight-shaped orbit, discovered just recently (2000) by Chenciner and Montgomery [8]. You can "feed" those equations into the Taylor Center, integrate them, draw the curves and play the motion in the real-time mode all in the same place.

The features of the future versions of the product will include everything above plus the following:

- It will come not only as the Taylor Center GUI executable, but also as the separate units, allowing the users to include them directly in their Delphi projects to perform massive computations with ODEs, and also as a DLL to use in other environments;

- It will implement the Merge procedure and a library of ODEs, defining a large variety of commonly used elementary functions. (Presently, the functions which are not in the allowed limited list may be used also – providing that the user declares the ODEs defining them and properly links them with the source;

- It will work for complex numbers, which will make it possible to study the solutions of ODEs in the complex plane in order to locate and explore their singularities;

- The application will be ported to Linux/Kylex;

- The set of the internal differentiation instructions will be translated into the machine code - to reach the highest possible speed for massive computations. (Meanwhile it is an emulator written in Delphi which runs these instructions). Also, it may be translated into instructions in Pascal, C or Fortran to be further compiled and linked with other applications;

- Integrating by a parameter or integrating boundary value problems [6].

References

1. Moore, R. E.: Interval Analysis. Prenitce-Hall, Englewood Cliffs, N.Y. (1966).
2. Chang Y. F., The ATOMFT User Manual. Version 2.50, 976 W. Foothill Blvd . #298, Claremont, CA 91711 (1988)
3. Gofen, A. M.: Taylor Method of Integrating Ordinary Differential Equations: the Problem of Steps and Singularities. Cosmic Research, Vol. 30, No. 6, pp. 581-593 (1992)
4. Gofen, A. M.: Recursion Excursion. Delphi Informant Magazine. Vol. 6, No. 8 (2000)
5. Gofen, A. M.: Recursive Journey to Three Bodies. Delphi Informant Magazine. Vol. 8, No. 3 (2002)
6. Gofen, A. M.: ODEs and Redefining the Concept of Elementary Functions. These Proceedings (2002)
7. Gofen, A. M.: The Taylor Center Demo for PCs. http://www.ski.org/rehab/mackeben/gofen/TaylorMethod.htm
8. Chenciner, Montgomery. The N-body problem "Choreography" www.maia.ub.es/dsg