INTEGRATOR: A Computational Tool to Solve Ordinary Differential Equations with Global Error Control

Gennady Y. Kulikov and Sergey K. Shindin

School of Computational and Applied Mathematics, University of the Witwatersrand, Private Bag 3, Wits 2050, Johannesburg, South Africa
{gkulikov,sshindin}@cam.wits.ac.za

Abstract. In recent papers [6]–[10] the technique for local and global errors estimation and the local-global step size control have been presented to solve both ordinary differential equations and semi-explicit index 1 differential-algebraic systems by multistep methods with any automatically obtained reasonable accuracy. Now we describe the object oriented library INTEGRATOR (ver. 1.0) built in C++ for portability and performance across a wide class of machine architectures. Also we give some computational tests and a comparison with the software implemented the well-known Gear's method which has been included into MATLAB (ver. 6.2, release 12).

The problem of an automatic global error control for ODEs of the form

$$x'(t) = g\big(t, x(t)\big), \quad t \in [t_0, t_0 + T], \quad x(t_0) = x^0, \tag{1}$$

where $x(t) \in \mathbf{R}^n$ and $g : D \subset \mathbf{R}^{n+1} \to \mathbf{R}^n$ is a sufficiently smooth function, is one of the challenges of modern computational mathematics. ODE (1) is quite usual in applied research and practical engineering (see, for example, [1]–[5]) Therefore any issue in that field possesses a great potential to develop intelligent software packages for mathematical modelling tasks. Thus, following the recent advances in computer algorithms for global error monitoring and controlling [7]–[10], we present the new user-oriented library INTEGRATOR for solving problem (1) with any accuracy set by the user (up to round-off). In this paper we give a brief outline of library types and routines as well as numerical experiments and data analysis. We clearly show an advantage of the new software at least over the implementation of Gear's method [3] taken from MATLAB (ver. 6.2, release 12) that is a good reason to use INTEGRATOR in practice.

Most of numerical algorithms imply vector and matrix computations in this or that form. Therefore, when constructing any object-oriented software, it is important to separate the algorithm from the vector-matrix arithmetic implementation for a convenience to use or to modify the code. Following this philosophy, we divided the library into two parts. The first one contains basic `C++` classes

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3037, pp. 570–573, 2004.

and routines providing facilities to code complex vectors-matrices algorithms closer to its natural mathematical notation. The second part includes routines implemented both multistep methods (with fixed or variable coefficients) and Nordsieck ones with the local-global step size control in order to solve ODEs of the form (1) with the accuracy set by the user.

Here, we do not describe the details concerning the vector and matrix classes because they are not essential for using INTEGRATOR in practice. We start with the common description of function-solvers implemented in the second part of our library. Each of these functions are defined in the standard way:

```
statistics <solver>(odestruct& _ODE, solveropt& _OPT);
```

The data structures `odestruct`, `solveropt` and `statistics` are declared in the header file `numerics.h` and contain input and output parameters of the function-solvers.

The input data structure `odestruct` has been designed to keep the input information about ODE (1) and it contains a pointer to the functions for evaluating $g(t, x)$ and its Jacobian $\partial_x g(t, x)$, a vector-column of the initial values x_0 for problem (1), and also the initial and final time points of the integration interval $[t_0, t_0 + T]$.

The input data structure `solveropt` includes four groups of parameters needed for the function-solvers. The first one is aimed to set the order of multistep method, the type of iteration, and the number of iteration steps per grid point. At present INTEGRATOR supports the following three iterative methods: `solveropt::mt_NEWTON` is a full Newton iteration, `solveropt::mt_mNEWTON` is a modified Newton iteration, `solveropt::mt_sITER` is a simple (or fixed-point) iteration. The lower bounds of enough iteration steps for each type of the iterative methods can be found in [7].

The second group of parameters is necessary to set particulars of the local-global step size control (the detailed description of that algorithm for multistep methods with both fixed coefficients and variable ones applied to ordinary differential equations or to differential-algebraic systems can be found in [6]–[10]). In particular, these parameters include the global error tolerance ϵ_g, the safety factor, the initial step size τ_1 for an starting procedure, and the initial restriction for the maximum step size τ.

The third group of parameters as well as the fourth one manages an output of statistical and integration data to the consol and the user defined `C++` streams, respectively.

All function-solvers return the data structure `statistics`. The data fields of this structure preserve the statistical data accumulated during the integration process of problem (1).

The header files `bdf.h` and `adams.h` declare all the function-solvers of the library. The first one contains algorithms based on Backward Differentiation Formulas (BDF methods): the function-solver `VBDF_SCL` implements fixed-coefficients BDF methods with polynomial interpolation of the numerical solution; the function-solver `VBDF_SCH` implements variable-coefficients BDF methods; the function-solver `NBDF_SCH` implements the Nordsieck representation for

BDF methods. The library INTEGRATOR supports the BDF methods of orders 2–6.

Table 1. Global errors obtained when treating problem (0.1) by the fixed-coefficients implicit Adams methods (with extrapolation) according to order s

order	required accuracy				
s	$\epsilon_g = 10^{-02}$	$\epsilon_g = 10^{-03}$	$\epsilon_g = 10^{-04}$	$\epsilon_g = 10^{-05}$	$\epsilon_g = 10^{-06}$
2	$2.11 \cdot 10^{-05}$	$2.42 \cdot 10^{-07}$	—	—	—
3	$3.15 \cdot 10^{-04}$	$1.08 \cdot 10^{-05}$	$5.03 \cdot 10^{-07}$	$2.01 \cdot 10^{-08}$	$1.14 \cdot 10^{-09}$
4	$5.41 \cdot 10^{-05}$	$2.70 \cdot 10^{-06}$	$1.69 \cdot 10^{-07}$	$8.70 \cdot 10^{-09}$	$3.18 \cdot 10^{-10}$
5	$1.26 \cdot 10^{-04}$	$1.94 \cdot 10^{-05}$	$2.68 \cdot 10^{-06}$	$2.46 \cdot 10^{-07}$	$2.68 \cdot 10^{-08}$
6	$4.11 \cdot 10^{-03}$	$1.75 \cdot 10^{-05}$	$8.49 \cdot 10^{-07}$	$5.60 \cdot 10^{-08}$	$3.37 \cdot 10^{-09}$
7	$1.76 \cdot 10^{-04}$	$7.60 \cdot 10^{-06}$	$8.82 \cdot 10^{-08}$	$2.80 \cdot 10^{-08}$	$4.00 \cdot 10^{-09}$

Table 2. Global errors obtained when treating problem (1.20) by the variable-coefficients BDF methods (with extrapolation) according to order s

order	required accuracy				
s	$\epsilon_g = 10^{-02}$	$\epsilon_g = 10^{-03}$	$\epsilon_g = 10^{-04}$	$\epsilon_g = 10^{-05}$	$\epsilon_g = 10^{-06}$
2	$2.54 \cdot 10^{-06}$	$3.08 \cdot 10^{-08}$	$7.53 \cdot 10^{-09}$	$8.19 \cdot 10^{-11}$	$3.15 \cdot 10^{-08}$
3	$1.11 \cdot 10^{-05}$	$6.57 \cdot 10^{-07}$	$2.38 \cdot 10^{-08}$	$1.83 \cdot 10^{-09}$	$4.16 \cdot 10^{-10}$
4	$5.40 \cdot 10^{-05}$	$2.96 \cdot 10^{-06}$	$1.72 \cdot 10^{-07}$	$1.35 \cdot 10^{-08}$	$5.87 \cdot 10^{-10}$
5	$6.03 \cdot 10^{-04}$	$4.24 \cdot 10^{-05}$	$2.99 \cdot 10^{-06}$	$1.44 \cdot 10^{-07}$	$8.30 \cdot 10^{-09}$
6	$2.64 \cdot 10^{-04}$	$7.12 \cdot 10^{-05}$	$3.37 \cdot 10^{-06}$	$2.85 \cdot 10^{-07}$	$2.03 \cdot 10^{-08}$

The second header file includes algorithms based on implicit Adams methods: the function-solver `VADM_SCL` implements fixed-coefficients implicit Adams methods with polynomial interpolation of the numerical solution; the function-solver `VADM_SCH` implements variable-coefficients implicit Adams methods; the function-solver `NADM_SCH` implements the Nordsieck representation for implicit Adams methods. Orders of all the Adams methods have to be from 2 up to 7.

Finally, we give a couple of numerical examples showing the efficiency of INTEGRATOR. As test problems, we take ODEs (0.1) and (1.20) from [4]. The first one is the restricted three body problem and possesses the periodic solution-path. The second ODE has the exact solution. Thus, we are capable to observe the work of our software in practice.

Now we apply the function-solvers of INTEGRATOR described above to these test problems. We use both Adams methods and BDF ones with fixed and variable coefficients to compute the numerical solutions. We also determine the real errors appeared in the integrations and compare them with the required accuracy. The lines in Table 1 mean that the second order function-solvers are not able to find the numerical solution with the set accuracy when $\epsilon_g \leq 10^{-04}$ because the required step size in this situation is smaller than the minimum admissible step size value $\tau_{\min}$.

Table 3. Global errors obtained when treating the test problems by the Gear's method from MATLAB (ver. 6.2, release 12)

test problem	required accuracy $\epsilon_l = 10^{-02}$	$\epsilon_l = 10^{-03}$	$\epsilon_l = 10^{-04}$	$\epsilon_l = 10^{-05}$	$\epsilon_l = 10^{-06}$	$\epsilon_l = 10^{-07}$
(0.1)	$2.09 \cdot 10^{+00}$	$2.07 \cdot 10^{+00}$	$1.94 \cdot 10^{+00}$	$1.45 \cdot 10^{+00}$	$6.00 \cdot 10^{-02}$	$1.00 \cdot 10^{-03}$
(1.20)	$1.40 \cdot 10^{+02}$	$1.09 \cdot 10^{+02}$	$9.50 \cdot 10^{+01}$	$2.15 \cdot 10^{+00}$	$2.92 \cdot 10^{-01}$	$6.23 \cdot 10^{-02}$

Tables 1, 2 display that all the function-solvers have achieved the goal; i.e., they have computed the numerical solutions of problems (0.1) and (1.20) with the set accuracy ϵ_g. To emphasis the advantage of INTEGRATOR over the standard software packages, we applied the Gear's method from MATLAB (ver. 6.2, release 12) to both test problems (see Table 3). A comparison of these data with Tables 1, 2 gives a right to conclude that the new software, in fact, controls automatically the global error of numerical solution. This is a good result for future testing of the library INTEGRATOR and for implementing it in practice.

References

1. Arushanyan, O.B., Zaletkin, S.F.: Numerical solution of ordinary differential equations using FORTRAN. (*in Russian*) Mosk. Gos. Univ., Moscow, 1990
2. Butcher, J.C.: Numerical methods for ordinary differential equations. John Wiley & Son, Chichester, 2003
3. Gear, C.W.: Numerical initial value problems in ordinary differential equations. Prentice-Hall, 1971
4. Hairer, E., Nørsett, S.P., Wanner, G.: Solving ordinary differential equations I: Nonstiff problems. Springer-Verlag, Berlin, 1987
5. Hairer, E., Wanner, G.: Solving ordinary differential equations II: Stiff and differential-algebraic problems. Springer-Verlag, Berlin, 1996
6. Kulikov, G.Yu., Shindin, S.K.: A local-global stepsize control for multistep methods applied to semi-explicit index 1 differential-algebraic equations. Korean J. Comput. Appl. Math. **6** (1999) No. 3, 463–492
7. Kulikov, G.Yu., Shindin, S.K.: A technique for controlling the global error in multistep methods. (*in Russian*) Zh. Vychisl. Mat. Mat. Fiz. **40** (2000) No. 9, 1308–1329; *translation in* Comput. Math. Math. Phys. **40** (2000) No. 9, 1255–1275
8. Kulikov, G.Yu., Shindin, S.K.: On multistep extrapolation methods for ordinary differential equations. (*in Russian*) Dokl. Akad. Nauk, **372** (2000) No. 3, 301–304; *translation in* Doklady Mathematics, **61** (2000) No. 3, 357–360
9. Kulikov, G.Yu., Shindin, S.K.: On effective computation of asymptotically correct estimates of the local and global errors for multistep methods with fixed coefficients. (*in Russian*) Zh. Vychisl. Mat. Mat. Fiz. (to appear); *translation in* Comput. Math. Math. Phys. (to appear)
10. Kulikov, G.Yu., Shindin, S.K.: On interpolation type multistep methods with automatic global error control. (*in Russian*) Zh. Vychisl. Mat. Mat. Fiz. (to appear); *translation in* Comput. Math. Math. Phys. (to appear)