Comparing ODE Solvers for to ARGESIM Benchmark C5 'Two State Model' using MATLAB

Florian Judex, Florian Plug, Ido Yehieli, Vienna University of Technology, Austria, *efelo@fsmat.at*

Simulator: Matlab is a well-known general-purpose mathematical programming language. It offers a wealth of predefined functions especially suited for implementing numerical algorithms. One of those is `ode15s`, which implements a BDF method for computing numerical solutions of stiff ordinary differential equations and supports event detection/location. We compared solutions of C5 obtained with `ode15s` with those obtained using a handwritten solver using an implicit Runge-Kutta scheme based on Gauss-Legendre quadrature and bisection (for event location). Since C5 is easily solved analytically (either manually, or using a computer algebra system like Maple), we also compared those solutions to an analytically obtained solution to judge their accuracy.

Model: Comparison 5 is a simple system of two linear ordinary differential equations, given by

$$\dot{y} = \begin{pmatrix} -c_1 & c_1 \\ 0 & -c_3 \end{pmatrix} \cdot y + \begin{pmatrix} c_1 \cdot c_2 \\ c_3 \cdot c_4 \end{pmatrix} \quad (1)$$

Parameters c_2 and c_4 depend on the current state of the System (which can either be A or B), while c_1 and c_3 are kept constant over the whole integration interval. The state is determined by two bounds Y_A and Y_B – whenever y_1 grows larger than Y_A in state A, the system switches to state B, and conversely when y_1 becomes smaller than Y_B in state B the system switches back to state A. The interesting cases arise for values of c_2 and c_4 which produce periodic state changes between A and B, and values of c_1 and c_3 which turn the system into a stiff one.

`ode15s` operates on systems of ODEs given in the form $\dot{y}(t) = f(t, y(t))$ together with zero or more event functions $e_i(t,y)$. While integrating, the algorithms monitors the event functions, and locates their zeros-crossings. Two flags per event function specify if only rising, only falling, or both directions of zero-crossing shall be considered an event, and if the integration is to be continued or stopped upon detecting an event. The step size is controlled by a local error target passed to the algorithms. For more efficient and accurate operation, the jacobian $\partial f / \partial y$ of $f(y,t)$ can be specified.

A-Task: Comparison 5 is easily brought into the form required by `ode15s`. The definitions of $f(y,t)$ and $\partial f / \partial y$ of are obvious, and for state-change detection the two event functions $e_A(t,y) = y_1 - Y_A$ and $e_B(t,y) = Y_B - y_1$ can be used – both set to consider only rising zero-crossings. Because the description of `ode15s` is not entirely clear on how the algorithm continues after a state change, both functions were set to stop integration upon event detection, and `ode15s` was called in a loop until the whole integration interval was covered.

The implemented algorithm treats state changes differently – it requires the system of ODEs to be given as $(\dot{y}(t), s(t)) = f(t, y(t), s(t), p)$, with $f(t,y,s,p)$ linear in y (It could be extended to support nonlinear systems rather easily, though) and $s(t)$ specifying the current state of the system. The value for p is specified when calling the solver algorithm, and it passed down to the individual evaluations f. This allows parameterization of the system of ODEs without resorting to global variables. $f(t,y,s,p)$ must not only calculate the derivative of y, but also the new state of the system. Whenever the returned value of $s(t)$ differs from the one that was passed to $f(t,y,s,p)$, the algorithms treats this as a state event. The algorithm takes another argument specifying a global error target that it tries to meet.

The author's algorithm integrates along the integration interval, at each step controlling the step size by computing a local error estimate and comparing it to some local error target. When it encounters a state change event ($s(t)$ changes), it locates the precise time of the state change using bisection. It then recomputes the solution starting from the last state change (or the start of the integration interval) on a twice as fine grid, and computes a global error estimate by comparing the two solutions. If this estimate meets the requested global error target, it proceeds by restarting the integration using the last computed $y(t)$ and $s(t)$ as the new initial values. If the estimate doesn't meet the global error target, the integration restarts at the last state change, with a suitably reduced local error target.

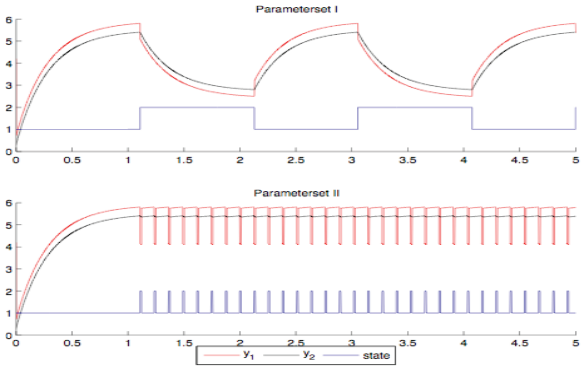

Figure 1. Plot of the solution

Each step is computed using a fully implicit 4-state Runge-Kutta scheme to accommodate the stiffness of the problem. The coefficients for that 4-state IRK scheme were obtained by Gauss-Legendre quadrature, and therefore yield an 8^{th} order scheme.

Figure 1 shows the results obtained with the implemented solver.

B-Task: For better results, the analytical solution was used to compute the zero Crossings given in Table 1.

	Analytical
t_1	1.1083061677711285586
t_2	2.1296853551547112460
t_3	3.0541529069957142895
t_4	4.0755320943792971988
t_5	4.9999996462203002423
$y_1(5)$	5.3693121180964613615
$y_2(5)$	5.3999967644598712013

Table 1. Zero Crossings

C-Task: Since only our handwritten solver supports global error estimation, the accuracy was interpreted as the global error target for our handwritten solver, but at the local error target for `ode15s`.

Table 2 shows the results produced by `do_test` for three different accuracies (10^{-6}, 10^{-10} and 10^{-14}).

D-Task: Using the same test setup as for the tasks b and c also the results for the other set of parameters werce computed, yielding a much higher oscillation frequency. The results of this experiment can be found in Table 3.

	10^{-6}		**10^{-10}**		**10^{-14}**	
	IRK (rel. err.)	ode15s (rel.err.)	IRK (rel. err.)	ode15s (rel.err.)	IRK (rel. err.)	ode15s (rel.err.)
t_1	4e-14	2e-9	4e-15	2e-9	7e-15	2e-9
t_2	2e-14	7e-7	1e-15	4e-9	2e-15	4e-9
t_3	6e-15	5e-7	3e-16	6e-9	2e-16	6e-9
t_4	5e-15	7e-7	1e-15	6e-9	2e-16	6e-9
t_5	4e-15	6e-7	1e-15	7e-9	5e-16	7e-9
$y_1(5)$	4e-9	5e-2	9e-10	4e-3	4e-10	4e-3
$y_2(5)$	3e-14	5e-6	1e-14	6e-8	5e-15	7e-9

Table 2: Accuracy of the Algorithms

		10^{-11}	
	Analytical	IRK (rel. err.)	ode15s (rel.err.)
t_1	1.108306167	1e-14	2e-9
t_2	1.121729967	1e-14	2e-7
t_{n-2}	4.809306109	5e-14	2e-6
t_{n-1}	4.923040107	5e-14	2e-6
t_n	4.936463907	5e-14	2e-6
$Y_1(5)$	5.780402520	2e-14	6e-7
$Y_2(5)$	5.380402678	2e-14	9e-7

Table 3: Results for Task D

These results show a similar relationship between the errors produces by ode15s compared to those of our handwritten algorithmn as tasks b and c. Since the last state change (t_N in the results table) lays further away from 5 for this set of parameters, $\dot{y}$ returns to a smaller value before reaching the end of the integration interval. This eliminates the large difference between the relative errors of $y_1(5)$ and $y_2(5)$.

Resumé: The implemented IRK algorithm fulfilled the authors expectations fully. It only remains to be implemented according to the MATLAB standard for ODE solvers.

Corresponding author: Florian Judex,
Department of Analysis and Scientific Computing
Vienna University of Technology
Wiedner Hauptstraße 8-10, 1040 Vienna, Austria
efelo@fsmat.at

Received: June 28, 2007
Revised: November 17, 2007
Revised: March 3, 2008
Accepted: March 10, 2008