Mathematical Tripos Part IB: Lent 2019

Numerical Analysis – Lecture 11[1]

4.6 Implementation of ODE methods

The step size h is not some preordained quantity: it is a parameter of the method (in reality, many parameters, since we may vary it from step to step). The basic input of a well-written computer package for ODEs is not the step size but the *error tolerance:* the level of precision, as required by the user. The choice of $h > 0$ is an important tool at our disposal to keep a local estimate of the error beneath the required tolerance in the solution interval. In other words, we need not just a *time-stepping algorithm,* but also mechanisms for *error control* and for amending the step size.

The Milne device Suppose that we wish to monitor the error of the trapezoidal rule

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \tfrac{1}{2}h[\boldsymbol{f}(t_n, \boldsymbol{y}_n) + \boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1})]. \tag{4.12}$$

We already know that the order is 2. Moreover, substituting the true solution we deduce that

$$\boldsymbol{y}(t_{n+1}) - \{\boldsymbol{y}(t_n) + \tfrac{1}{2}h[\boldsymbol{y}'(t_n) + \boldsymbol{y}'(t_{n+1})]\} = -\tfrac{1}{12}h^3\boldsymbol{y}'''(t_n) + \mathcal{O}\big(h^4\big)\,.$$

Therefore, the error in each step is increased roughly by $-\frac{1}{12}h^3\boldsymbol{y}'''(t_n)$. The number $c_{\mathrm{TR}} = -\frac{1}{12}$ is called the *error constant* of TR. Similarly, each multistep method (but not RK!) has its own error constant. For example, the 2nd order 2-step Adams–Bashforth method

$$\boldsymbol{y}_{n+1} - \boldsymbol{y}_n = \tfrac{1}{2}h[3\boldsymbol{f}(t_n, \boldsymbol{y}_n) - \boldsymbol{f}(t_{n-1}, \boldsymbol{y}_{n-1})], \tag{4.13}$$

has the error constant $c_{\mathrm{AB}} = \frac{5}{12}$.

The idea behind the *Milne device* is to use two multistep methods of the same order, one explicit and the second implicit (e.g., (4.13) and (4.12), respectively), to estimate the local error of the implicit method. For example, *locally,*

$$\begin{aligned}\boldsymbol{y}_{n+1}^{\mathrm{AB}} &\approx \boldsymbol{y}(t_{n+1}) - c_{\mathrm{AB}}h^3\boldsymbol{y}'''(t_n) = \boldsymbol{y}(t_{n+1}) - \tfrac{5}{12}h^3\boldsymbol{y}'''(t_n),\\ \boldsymbol{y}_{n+1}^{\mathrm{TR}} &\approx \boldsymbol{y}(t_{n+1}) - c_{\mathrm{TR}}h^3\boldsymbol{y}'''(t_n) = \boldsymbol{y}(t_{n+1}) + \tfrac{1}{12}h^3\boldsymbol{y}'''(t_n).\end{aligned}$$

Subtracting, we obtain the estimate $h^3\boldsymbol{y}'''(t_n) \approx -2(\boldsymbol{y}_{n+1}^{\mathrm{AB}} - \boldsymbol{y}_{n+1}^{\mathrm{TR}})$, therefore

$$\boldsymbol{y}_{n+1}^{\mathrm{TR}} - \boldsymbol{y}(t_{n+1}) \approx \tfrac{1}{6}(\boldsymbol{y}_{n+1}^{\mathrm{TR}} - \boldsymbol{y}_{n+1}^{\mathrm{AB}})$$

and we use the right hand side as an estimate of the local error.

Note that TR is a far better method than AB: it is A-stable, hence its *global* behaviour is superior. We employ AB *solely* to estimate the local error. This adds very little to the overall cost of TR, since AB is an explicit method.

Implementation of the Milne device We work with a *pair* of multistep methods of the same order, one explicit *(predictor)* and the other implicit *(corrector),* e.g.

$$\begin{aligned}\text{Predictor :}\quad & \boldsymbol{y}_{n+2} = \boldsymbol{y}_{n+1} + h[\tfrac{5}{12}\boldsymbol{f}(t_{n-1}, \boldsymbol{y}_{n-1}) - \tfrac{4}{3}\boldsymbol{f}(t_n, \boldsymbol{y}_n) + \tfrac{23}{12}\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1})],\\ \text{Corrector :}\quad & \boldsymbol{y}_{n+2} = \boldsymbol{y}_{n+1} + h[-\tfrac{1}{12}\boldsymbol{f}(t_n, \boldsymbol{y}_n) + \tfrac{2}{3}\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1}) + \tfrac{5}{12}\boldsymbol{f}(t_{n+2}, \boldsymbol{y}_{n+2})],\end{aligned}$$

the third-order Adams–Bashforth and Adams–Moulton methods respectively.

The predictor is employed not just to estimate the error of the corrector, but also to provide *an initial guess in the solution of the implicit corrector equations.* Typically, for nonstiff equations, we iterate correction equations at most twice, while stiff equations require *iteration to convergence*, otherwise the typically superior stability features of the corrector are lost.

[1] Corrections and suggestions to these notes should be emailed to `h.fawzi@damtp.cam.ac.uk`.

Let TOL > 0 be a user-specified *tolerance:* the maximal error allowed in approximating the ODE. Having completed a single step and estimated the error, there are three possibilities:

(a) $\frac{1}{10}\mathrm{TOL} \le \|\,\mathrm{error}\,\| \le \mathrm{TOL}$, say: Accept the step, continue to t_{n+2} with the same step size.

(b) $\|\,\mathrm{error}\,\| < \frac{1}{10}\mathrm{TOL}$, say: Accept the step and increase the step length;

(c) $\|\,\mathrm{error}\,\| > \mathrm{TOL}$: Reject the step, recommence integration from t_n with smaller h.

Amending step size can be done easily with polynomial interpolation, although this means that we need to store past values well in excess of what is necessary for simple implementation of both multistep methods.

Error estimation per unit step Let $\boldsymbol{e}$ be our estimate of *local* error. Then $\boldsymbol{e}/h$ is our estimate for the global error in an interval of unit length. It is usual to require the latter quantity not to exceed TOL since good implementations of numerical ODEs should monitor the accumulation of *global* error. This is called *error estimation per unit step.*

Embedded Runge–Kutta methods The situation is more complicated with RK, since no single error constant determines local growth of the error. The approach of *embedded RK* requires, again, two (typically explicit) methods: an RK method of ν stages and order p, say, and another method, of $\nu + l$ stages, $l \ge 1$, and order $p + 1$, such that *the first ν stages of both methods are identical.* (This means that the cost of implementing the higher-order method is marginal, once we have computed the lower-order approximation.) For example, consider (and verify!)

$$\begin{aligned}
\boldsymbol{k}_1 &= \boldsymbol{f}(t_n, \boldsymbol{y}_n),\\
\boldsymbol{k}_2 &= \boldsymbol{f}\left(t_n + \tfrac{1}{2}h, \boldsymbol{y}_n + \tfrac{1}{2}h\boldsymbol{k}_1\right),\\
\boldsymbol{y}_{n+1}^{[1]} &= \boldsymbol{y}_n + h\boldsymbol{k}_2 && \Longrightarrow \quad \text{order 2},\\
\boldsymbol{k}_3 &= \boldsymbol{f}(t_n + h, \boldsymbol{y}_n - h\boldsymbol{k}_1 + 2h\boldsymbol{k}_2),\\
\boldsymbol{y}_{n+1}^{[2]} &= \boldsymbol{y}_n + \tfrac{1}{6}h(\boldsymbol{k}_1 + 4\boldsymbol{k}_2 + \boldsymbol{k}_3) && \Longrightarrow \quad \text{order 3}.
\end{aligned}$$

We thus estimate $\boldsymbol{y}_{n+1}^{[1]} - \boldsymbol{y}(t_{n+1}) \approx \boldsymbol{y}_{n+1}^{[1]} - \boldsymbol{y}_{n+1}^{[2]}$. *[It might look paradoxical, at least at first glance, but the only purpose of the higher-order method is to provide error control for the lower-order one!]*

The Zadunaisky device Suppose that the ODE $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$, $\boldsymbol{y}(0) = \boldsymbol{y}_0$, is solved by an arbitrary numerical method of order p and that we have stored (not necessarily equidistant) past solution values $\boldsymbol{y}_n, \boldsymbol{y}_{n-1}, \ldots, \boldsymbol{y}_{n-p}$. We form an interpolating pth degree polynomial (with vector coefficients) $\boldsymbol{d}$ such that $\boldsymbol{d}(t_{n-i}) = \boldsymbol{y}_{n-i}$, $i = 0, 1, \ldots, p$, and consider the differential equation

$$\boldsymbol{z}' = \boldsymbol{f}(t, \boldsymbol{z}) + \boldsymbol{d}'(t) - \boldsymbol{f}(t, \boldsymbol{d}), \qquad \boldsymbol{z}(t_n) = \boldsymbol{y}_n. \tag{4.14}$$

There are two important observations with regard to (4.14)

(1) Since $\boldsymbol{d}(t) - \boldsymbol{y}(t) = \mathcal{O}\big(h^{p+1}\big)$, the term $\boldsymbol{d}'(t) - \boldsymbol{f}(t, \boldsymbol{d})$ is usually small (because $\boldsymbol{y}'(t) - \boldsymbol{f}(t, \boldsymbol{y}(t)) \equiv \boldsymbol{0}$). Therefore, (4.14) is a small perturbation of the original ODE.

(2) The exact solution of (4.14) is known: $\boldsymbol{z}(t) = \boldsymbol{d}(t)$.

Now, having produced $\boldsymbol{y}_{n+1}$ with our numerical method, we proceed to evaluate $\boldsymbol{z}_{n+1}$ as well, *using exactly the same method and implementation details.* We then evaluate the error in $\boldsymbol{z}_{n+1}$, namely $\boldsymbol{z}_{n+1} - \boldsymbol{d}(t_{n+1})$, and use it as an estimate of the error in $\boldsymbol{y}_{n+1}$.

Solving nonlinear algebraic systems We have already observed that the implementation of an implicit ODE method, whether multistep or RK, requires the solution of (in general, nonlinear) algebraic equations in each step. For example, for an s-step method, we need to solve in each step the algebraic system

$$\boldsymbol{y}_{n+s} = \sigma_s h\boldsymbol{f}(t_{n+s}, \boldsymbol{y}_{n+s}) + \boldsymbol{v}, \tag{4.15}$$

where the vector $\boldsymbol{v}$ can be formed from past (hence known) solution values and their derivatives. The easiest approach is *functional iteration*

$$\boldsymbol{y}_{n+s}^{[j+1]} = \sigma_s h\boldsymbol{f}(t_{n+s}, \boldsymbol{y}_{n+s}^{[j]}) + \boldsymbol{v}, \qquad j = 0, 1, \ldots,$$

where $\boldsymbol{y}_{n+s}^{[0]}$ is typically provided by the predictor scheme. It is very effective for *nonstiff* equations but fails for *stiff ODEs*, since the convergence of this iterative scheme requires similar restriction on h as that we strive to avoid by choosing an implicit method in the first place!