Numerical Analysis – Lecture 14[1]

Implementation 3.37 To implement 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.

Let $\mathrm{TOL} > 0$ 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.

Improvement 3.38 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.*

Technique 3.39 *(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!)

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

We thus estimate $\boldsymbol{y}^{[1]}_{n+1} - \boldsymbol{y}(t_{n+1}) \approx \boldsymbol{y}^{[1]}_{n+1} - \boldsymbol{y}^{[2]}_{n+1}$. *[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!]*

Technique 3.39 *(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

[1]Please email all corrections and suggestions to these notes to `A.Iserles@damtp.cam.ac.uk`. All handouts are available on the WWW at the URL `http://www.damtp.cam.ac.uk/user/na/PartII/Handouts.html`.

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

$$\boldsymbol{z}' = \boldsymbol{f}(t, \boldsymbol{z}) + \boldsymbol{d}'(t) - \boldsymbol{f}(t, \boldsymbol{d}), \qquad \boldsymbol{z}(t_n) = \boldsymbol{y}_n. \tag{3.13}$$

There are two important observations with regard to (3.13)

(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, (3.13) is a small perturbation of the original ODE.

(2) The exact solution of (3.13) 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 $\boldsymbol{y}_{n+1}$.

Problem 3.40: 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

$$\boldsymbol{y}_{n+s} = \sigma_s h \boldsymbol{f}(t_{n+s}, \boldsymbol{y}_{n+s}) + \boldsymbol{v}, \tag{3.14}$$

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!

If the ODE is stiff, we might prefer a *Newton–Raphson* method, namely

$$\boldsymbol{y}_{n+s}^{[j+1]} = \boldsymbol{y}_{n+s}^{[j]} - \left[I - \sigma_s h \frac{\partial \boldsymbol{f}(t_{n+s}, \boldsymbol{y}_{n+s}^{[j]})}{\partial \boldsymbol{y}} \right]^{-1} [\boldsymbol{y}_{n+s}^{[j]} - \sigma_s h \boldsymbol{f}(t_{n+s}, \boldsymbol{y}_{n+s}^{[j]}) - \boldsymbol{v}].$$

The justification of the above is as follows: suppose that $y_{n+s}^{[j]}$ is an approximation to the solution. We linearise (3.14) locally about $(t_{n+s}, \boldsymbol{y}_{n+s}^{[j]})$,

$$\boldsymbol{y}_{n+s} - \sigma_s h \boldsymbol{f}(t_{n+s}, \boldsymbol{y}_{n+s}) - \boldsymbol{v} \approx [\boldsymbol{y}_{n+s}^{[j]} - \sigma_s h \boldsymbol{f}(t_{n+s}, \boldsymbol{y}_{n+s}^{[j]}) - \boldsymbol{v}] + \left[I - \sigma_s h \frac{\partial \boldsymbol{f}(t_{n+s}, \boldsymbol{y}_{n+s}^{[j]})}{\partial \boldsymbol{y}} \right] (\boldsymbol{y}_{n+s} - \boldsymbol{y}_{n+s}^{[j]})$$

and choose $\boldsymbol{y}_{n+s}^{[j+1]}$ by equating the right-hand side to $\boldsymbol{0}$.

The snag is that repeatedly evaluating and inverting (i.e. LU-factorizing) the Jacobian matrix in every iteration is *very* expensive. The remedy is to implement the *modified Newton–Raphson method*, namely

$$\boldsymbol{y}_{n+s}^{[j+1]} = \boldsymbol{y}_{n+s}^{[j]} - \left[I - \sigma_s h \frac{\partial \boldsymbol{f}(t_{n+s}, \boldsymbol{y}_{n+s}^{[0]})}{\partial \boldsymbol{y}} \right]^{-1} [\boldsymbol{y}_{n+s}^{[j]} - \sigma_s h \boldsymbol{f}(t_{n+s}, \boldsymbol{y}_{n+s}^{[j]}) - \boldsymbol{v}]. \tag{3.15}$$

Thus, the Jacobian need be evaluated only *once* a step.

The only role the Jacobian matrix plays in (3.15) is to ensure convergence: its precise value makes no difference to the ultimate value of $\lim_{j\to\infty} \boldsymbol{y}_{n+s}^{[j]}$. Therefore we might replace it with a finite-difference approximation, evaluate it once every several steps etc. Further note that the implementation of the method requires the LU factorization of $I - \sigma_s h J$, where J is our approximation of the Jacobian matrix, only once we change either h or J. Recalling that, once the LU factorization is known, the solution of linear systems is cheaper by an order of magnitude than the LU factorization itself, we deduce that the overall cost of this procedure is not excessive. For stiff equations it is definitely cheaper than using a minute step size with a 'bad' (e.g., explicit multistep or explicit RK) method.